**DAA CODES**

//Merge Sort

#include <stdio.h>

#include <stdlib.h>

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

void merge\_sort(int arr[], int l, int r)

{

if (l < r) {

int m = l + (r - l) / 2;

merge\_sort(arr, l, m);

merge\_sort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void printArray(int arr[], int size)

{

int i;

for (i = 0; i < size; i++)

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

printf("\n");

}

int main()

{

int arr[] = { 15,5,24,8,1,3,16,10,20 };

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

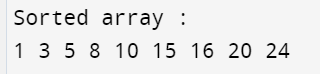
merge\_sort(arr, 0, n - 1);

printf("Sorted array : \n");

printArray(arr, n);

return 0;

}



//Quick Sort

#include <stdio.h>

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

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition(int arr[], int l, int h)

{

int x = arr[h];

int i = (l - 1);

int j;

for(j = l; j <= h - 1; j++)

{

if (arr[j] <= x)

{

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[h]);

return (i + 1);

}

void quick\_sort(int arr[], int l, int h)

{

int stack[h - l + 1];

int top = -1;

stack[++top] = l;

stack[++top] = h;

while (top >= 0)

{

h = stack[top--];

l = stack[top--];

int p = partition(arr, l, h);

if (p - 1 > l)

{

stack[++top] = l;

stack[++top] = p - 1;

}

if (p + 1 < h)

{

stack[++top] = p + 1;

stack[++top] = h;

}

}

}

void printArray(int arr[], int n)

{

int i;

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

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

}

int main()

{

int arr[] = {7,6,10,5,9,2,1,15,7 };

int n = sizeof(arr) / sizeof(\*arr);

quick\_sort(arr, 0, n - 1);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}



//Heap Sort

#include <stdio.h>

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

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void heapify(int arr[], int n, int i)

{

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && arr[left] > arr[largest])

largest = left;

if (right < n && arr[right] > arr[largest])

largest = right;

if (largest != i)

{

swap(&arr[i], &arr[largest]);

heapify(arr, n, largest);

}

}

void heap\_sort(int arr[], int n)

{

int i;

for ( i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

for (i = n - 1; i >= 0; i--)

{

swap(&arr[0], &arr[i]);

heapify(arr, i, 0);

}

}

void printArray(int arr[], int n)

{

int i;

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

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

printf("\n");

}

int main()

{

int arr[] = { 15,20,7,9,30 };

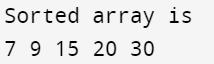
int n = sizeof(arr) / sizeof(arr[0]);

heap\_sort(arr, n);

printf("Sorted array is\n");

printArray(arr, n);

}



//Fractional Knapsack

#include <stdio.h>

int n = 5;

int p[10] = {3, 3, 2, 5, 1};

int w[10] = {10, 15, 10, 12, 8};

int W = 10;

int main(){

int cur\_w;

float tot\_v;

int i, maxi;

int used[10];

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

used[i] = 0;

cur\_w = W;

while (cur\_w > 0) {

maxi = -1;

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

if ((used[i] == 0) &&

((maxi == -1) || ((float)w[i]/p[i] > (float)w[maxi]/p[maxi])))

maxi = i;

used[maxi] = 1;

cur\_w -= p[maxi];

tot\_v += w[maxi];

if (cur\_w >= 0)

printf("Added object %d (%d, %d) completely in the bag. Space left: %d.\n", maxi + 1, w[maxi], p[maxi], cur\_w);

else {

printf("Added %d%% (%d, %d) of object %d in the bag.\n", (int)((1 + (float)cur\_w/p[maxi]) \* 100), w[maxi], p[maxi], maxi + 1);

tot\_v -= w[maxi];

tot\_v += (1 + (float)cur\_w/p[maxi]) \* w[maxi];

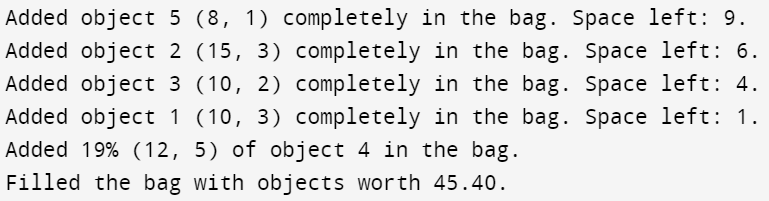
}

}

printf("Filled the bag with objects worth %.2f.\n", tot\_v);

return 0;

}



//Job Sequencing with Deadlines

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

typedef struct Job

{

char id;

int dead;

int profit;

} Job;

int compare(const void\* a, const void\* b)

{

Job\* temp1 = (Job\*)a;

Job\* temp2 = (Job\*)b;

return (temp2->profit - temp1->profit);

}

int min(int num1, int num2)

{

return (num1 > num2) ? num2 : num1;

}

void printJobScheduling(Job arr[], int n)

{

qsort(arr, n, sizeof(Job), compare);

int result[n];

bool slot[n];

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

slot[i] = false;

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

{

for (int j = min(n, arr[i].dead) - 1; j >= 0; j--)

{

if (slot[j] == false)

{

result[j] = i;

slot[j] = true;

break;

}

}

}

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

if (slot[i])

printf("%c ", arr[result[i]].id);

}

int main()

{

Job arr[] = { { 'a', 2, 100 },

{ 'b', 1, 19 },

{ 'c', 2, 27 },

{ 'd', 1, 25 },

{ 'e', 3, 15 } };

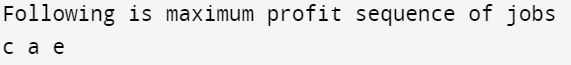
int n = sizeof(arr) / sizeof(arr[0]);

printf("Following is maximum profit sequence of jobs \n");

printJobScheduling(arr, n);

return 0;

}



//Dijkstra Algorithm

#include <stdio.h>

#define INFINITY 9999

#define MAX 10

void Dijkstra(int Graph[MAX][MAX], int n, int start)

{

int cost[MAX][MAX], distance[MAX], pre[MAX];

int visited[MAX], count, min\_distance, next\_node, i, j;

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

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

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

cost[i][j] = INFINITY;

else

cost[i][j] = Graph[i][j];

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

{

distance[i] = cost[start][i];

pre[i] = start;

visited[i] = 0;

}

distance[start] = 0;

visited[start] = 1;

count = 1;

while (count < n - 1) {

min\_distance = INFINITY;

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

if (distance[i] < min\_distance && !visited[i]) {

min\_distance = distance[i];

next\_node = i;

}

visited[next\_node] = 1;

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

if (!visited[i])

if (min\_distance + cost[next\_node][i] < distance[i])

{

distance[i] = min\_distance + cost[next\_node][i];

pre[i] = next\_node;

}

count++;

}

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

if (i != start)

{

printf("\nDistance from %d to %d: %d",start, i, distance[i]);

}

}

int main()

{

int Graph[MAX][MAX], i, j, n, s;

n = 4;

printf("Enter the source vertex: ");

scanf("%d",&s);

Graph[0][0] = 0;

Graph[0][1] = 0;

Graph[0][2] = 1;

Graph[0][3] = 2;

Graph[1][0] = 0;

Graph[1][1] = 0;

Graph[1][2] = 2;

Graph[1][3] = 0;

Graph[2][0] = 1;

Graph[2][1] = 2;

Graph[2][2] = 0;

Graph[2][3] = 1;

Graph[3][0] = 2;

Graph[3][1] = 0;

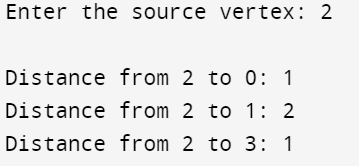
Graph[3][2] = 1;

Graph[3][3] = 0;

Dijkstra(Graph, n, s);

return 0;

}



//Bellman Ford Algorithm

#include <stdio.h>

#define INFINITY 9999

#define MAX 5

void bellmanFord(int graph[MAX][MAX], int src) {

int dist[MAX];

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

dist[i] = INFINITY;

dist[src] = 0;

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

{

for (int u = 0; u < MAX; u++)

{

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

{

if (graph[u][v] != 0 && dist[u] != INFINITY && dist[u] + graph[u][v] < dist[v]) {

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

}

}

}

}

for (int u = 0; u < MAX; u++)

{

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

{

if (graph[u][v] != 0 && dist[u] != INFINITY && dist[u] + graph[u][v] < dist[v])

{

printf("Graph contains negative-weight cycle\n");

return;

}

}

}

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

if(i!=src)

printf("\nDistance from %d to %d is %d ",src,i,dist[i]);

}

int main()

{

int Graph[MAX][MAX],n,s;

n=4;

printf("Enter the source: ");

scanf("%d",&s);

Graph[0][0]=0;

Graph[0][1]=-1;

Graph[0][2]=4;

Graph[0][3]=0;

Graph[1][0]=0;

Graph[1][1]=0;

Graph[1][2]=3;

Graph[1][3]=2;

Graph[2][0]=0;

Graph[2][1]=0;

Graph[2][2]=0;

Graph[2][3]=0;

Graph[3][0]=0;

Graph[3][1]=1;

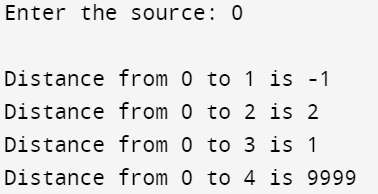
Graph[3][2]=5;

Graph[3][3]=0;

bellmanFord(Graph,s);

return 0;

}



//0-1 Knapsack

(Recursion method)

#include<stdio.h>

int max(int a, int b)

{

return (a > b)? a : b;

}

int knapSack(int W, int weight[], int profit[], int n)

{

if (n == 0 || W == 0)

return 0;

if (weight[n-1] > W)

return knapSack(W, weight, profit, n-1);

else

return max( profit[n-1] + knapSack(W-weight[n-1], weight, profit, n-1), knapSack(W, weight, profit, n-1));

}

int main()

{

int n,i,W;

printf("\nEnter the number of items : ");

scanf("%d", &n);

int profit[n];

int weight[n];

printf("\nEnter the item's profit and its weight \n");

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

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

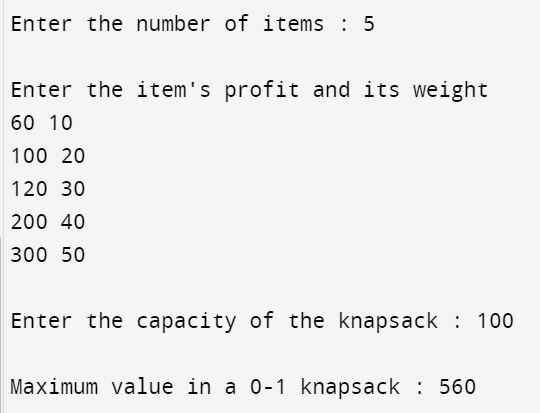
printf("\nEnter the capacity of the knapsack : ");

scanf("%d", &W);

printf("\nMaximum value in a 0-1 knapsack : %d\n", knapSack(W, weight,profit, n));

return 0;

}



(Tabulation method)

#include <stdio.h>

#define MAX\_WEIGHT 100

#define NUM\_ITEMS 5

int max(int a, int b)

{

return (a > b) ? a : b;

}

int main()

{

int values[] = {60, 100, 120, 200, 300};

int weights[] = {10, 20, 30, 40, 50};

int knapsack[NUM\_ITEMS + 1][MAX\_WEIGHT + 1];

int i,j;

for (i = 0; i <= NUM\_ITEMS; i++)

{

for (j = 0; j <= MAX\_WEIGHT; j++)

{

knapsack[i][j] = 0;

}

}

for (i = 1; i <= NUM\_ITEMS; i++)

{

for (j = 1; j <= MAX\_WEIGHT; j++)

{

if (weights[i - 1] > j)

{

knapsack[i][j] = knapsack[i - 1][j];

}

else

{

knapsack[i][j] = max(knapsack[i - 1][j], knapsack[i - 1][j - weights[i - 1]] + values[i - 1]);

}

}

}

printf("The optimal solution is: %d\n",knapsack[NUM\_ITEMS][MAX\_WEIGHT]);

}



//Matrix Chain Multiplication

#include <stdio.h>

#include<limits.h>

#define INFINITY 9999

int A[20][20],B[20][20],d[20],i,j,n;

void printOptimal(int i,int j)

{

if (i == j)

printf(" A%d ",i);

else

{

printf("( ");

printOptimal(i, B[i][j]);

printOptimal(B[i][j] + 1, j);

printf(" )");

}

}

void mcm()

{

int c,k;

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

{

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

{

if(i==j)

A[i][j]=0;

else

{

for(k=i;k<j;k++)

{

c=A[i][k]+B[k+1][j]+d[i-1]\*d[k]\*d[j];

if(c<A[i][j])

{

A[i][j]=c;

B[i][j]=k;

}

}

}

}

}

}

int mcOrder(int d[], int i, int j)

{

if(i == j)

return 0;

int k,min = INT\_MAX,count;

for (k = i; k <j; k++)

{

count = mcOrder(d, i, k) + mcOrder(d, k+1, j) +d[i-1]\*d[k]\*d[j];

if (count < min)

min = count;

}

return min;

}

int main()

{

int k;

printf("Enter the no. of elements: ");

scanf("%d",&n);

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

{

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

{

A[i][i]=0;

A[i][j]=INFINITY;

B[i][j]=0;

}

}

printf("\nEnter the dimensions: \n");

for(k=0;k<=n;k++)

{

printf("D%d: ",k);

scanf("%d",&d[k]);

}

mcm();

printf("\nCost Matrix M:\n");

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

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

printf("A[%d][%d]: %d\n",i,j,A[i][j]);

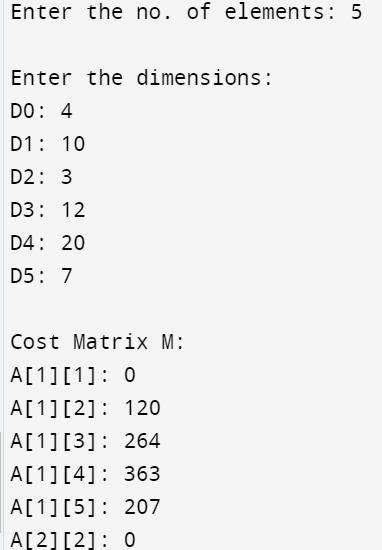
i=1,j=n;

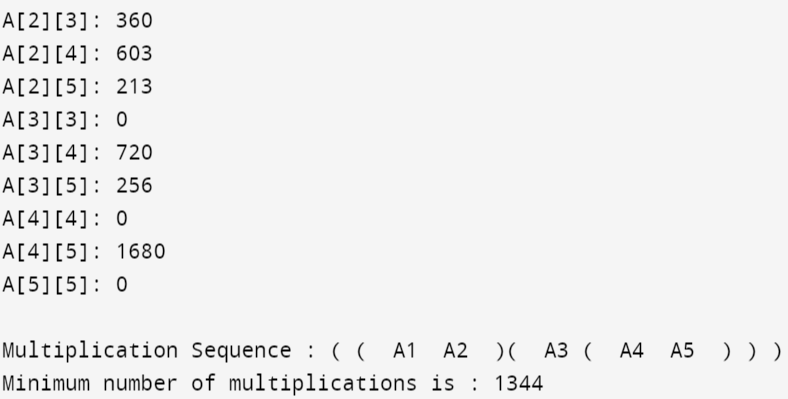
printf("\nMultiplication Sequence : ");

printOptimal(i,j);

printf("\nMinimum number of multiplications is : %d ",mcOrder(d, 1, n));

}





//Floyd Warshall Algorithm

#include<stdio.h>

int i,j,k;

void floydWarshall(int m[4][4], int n)

{

for(k=0;k<n;k++)

{

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

{

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

{

if(m[i][j]>m[i][k]+m[k][j])

{

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

}

}

}

}

printf("All Pairs Shortest Path is :\n");

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

{

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

{

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

}

printf("\n");

}

}

int main()

{

int cost[4][4] = {

{0, 3, 999, 999},

{999, 0, 12, 5},

{4, 999, 0, -1},

{2, -4, 999, 0}};

int n = 4;

floydWarshall(cost,n);

}

