1. Implement 0/1 Knapsack problem using dynamic programming.

#include <stdio.h>

int max(int a, int b) {

return (a > b) ? a : b;

}

int knapsack(int W, int wt[], int val[], int n) {

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

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

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

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

dp[i][w] = 0;

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

dp[i][w] = max(val[i-1] + dp[i-1][w - wt[i-1]], dp[i-1][w]);

else

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

}

}

return dp[n][W];

}

int main() {

int val[] = {60, 100, 120};

int wt[] = {10, 20, 30};

int W = 50;

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

printf("Maximum value in Knapsack = %d\n", knapsack(W, wt, val, n));

return 0;

}

Output:

A black background with white text

AI-generated content may be incorrect.

1. Write program to obtain the Topological ordering of vertices in a given digraph.

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

int adj[MAX][MAX], visited[MAX], stack[MAX], top = -1, n;

void dfs(int v) {

visited[v] = 1;

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

if (adj[v][i] && !visited[i])

dfs(i);

stack[++top] = v;

}

int main() {

int edges, u, v;

printf("Enter number of vertices: ");

scanf("%d", &n);

printf("Enter number of edges: ");

scanf("%d", &edges);

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

printf("Enter edge (u v): ");

scanf("%d %d", &u, &v);

adj[u][v] = 1;

}

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

if (!visited[i])

dfs(i);

printf("Topological order: ");

while (top >= 0)

printf("%d ", stack[top--]);

printf("\n");

return 0;

}

Output:

