

DAA Practical No 08

Implement Dynamic Algorithm for shortest path

1)Floyd Warshall Algorithm

2)Knapsack Algorithm

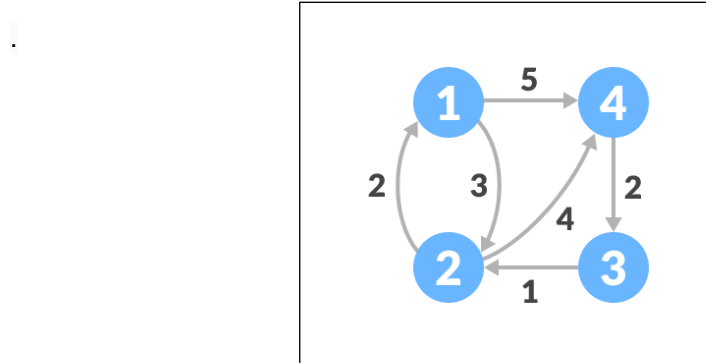
-->

1)Floyd Warshall Algorithm

Floyd-Warshall Algorithm is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph. This algorithm works for both the directed and undirected weighted graphs. But, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative)

This algorithm follows the dynamic programming approach to find the shortest paths

Let the given graph be:



Initial graph

Program Code:

```
#include <iostream>

using namespace std;

// defining the number of vertices

#define nV 4

#define INF 999

void printMatrix(int matrix[][nV]);

// Implementing floyd warshall algorithm

void floydWarshall(int graph[][nV]) {

    int matrix[nV][nV], i, j, k;

    for (i = 0; i < nV; i++)
```

```

    for (j = 0; j < nV; j++)
        matrix[i][j] = graph[i][j];

// Adding vertices individually
for (k = 0; k < nV; k++) {
    for (i = 0; i < nV; i++) {
        for (j = 0; j < nV; j++) {
            if (matrix[i][k] + matrix[k][j] < matrix[i][j])
                matrix[i][j] = matrix[i][k] + matrix[k][j];
        }
    }
}
printMatrix(matrix);
}

```

```

void printMatrix(int matrix[][nV]) {
    for (int i = 0; i < nV; i++) {
        for (int j = 0; j < nV; j++) {
            if (matrix[i][j] == INF)
                printf("%4s", "INF");
            else
                printf("%4d", matrix[i][j]);
        }
        printf("\n");
    }
}

```

```

int main() {
    int graph[nV][nV] = {{0, 3, INF, 5},
                          {2, 0, INF, 4},
                          {INF, 1, 0, INF},
                          {INF, INF, 2, 0}};

    floydWarshall(graph);
}

```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

● PS C:\Users\DELL\Desktop\DAA> cd 'c:\Users\DELL\Desktop\DAA\p8\output'
● PS C:\Users\DELL\Desktop\DAA\p8\output> & .\Floyd_warshall_Algo.exe
○   0   3   7   5
    2   0   6   4
    3   1   0   5
    5   3   2   0
PS C:\Users\DELL\Desktop\DAA\p8\output> █
```

2)Knapsack Algorithm:

What is the 0/1 Knapsack Problem?

We are given N items where each item has some weight and profit associated with it. We are also given a bag with capacity W, [i.e., the bag can hold at most W weight in it]. The target is to put the items into the bag such that the sum of profits associated with them is the maximum possible.

Note: The constraint here is we can either put an item completely into the bag or cannot put it at all [It is not possible to put a part of an item into the bag].

>>Program Code:

// A dynamic programming based solution for 0-1 Knapsack problem

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
// A utility function that returns maximum of two integers
```

```
int max(int a, int b){
```

```
    return (a > b) ? a : b;
```

```
}
```

```
// Returns the maximum value that can be put in a knapsack of capacity W
```

```
int knapSack(int W, int wt[], int val[], int n)
```

```
{
```

```
    int i, w;
```

```
    vector<vector<int>> > K(n + 1, vector<int>(W + 1));
```

```
    // Build table K[][] in bottom up manner
```

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

```

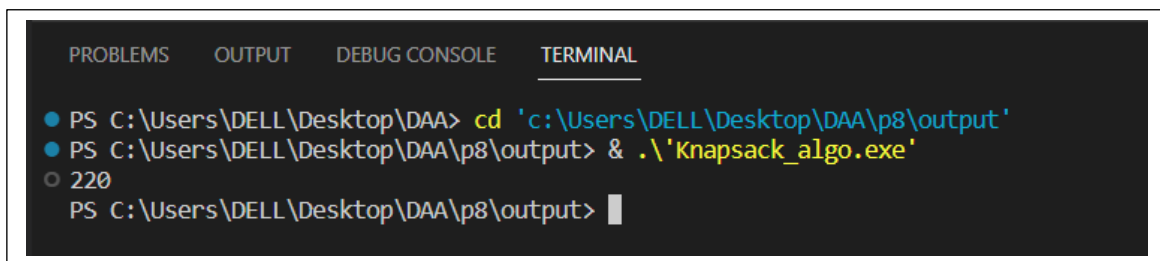
        for (w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                K[i][w] = 0;
            else if (wt[i - 1] <= w)
                K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);
            else
                K[i][w] = K[i - 1][w];
        }
    }

    return K[n][W];
}

// Driver Code
int main()
{
    int profit[] = { 60, 100, 120 };
    int weight[] = { 10, 20, 30 };
    int W = 50;
    int n = sizeof(profit) / sizeof(profit[0]);
    cout << knapSack(W, weight, profit, n);
    return 0;
}

```

Output:



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
● PS C:\Users\DELL\Desktop\DAA> cd 'c:\Users\DELL\Desktop\DAA\p8\output'
● PS C:\Users\DELL\Desktop\DAA\p8\output> & .\'knapSack_algo.exe'
○ 220
PS C:\Users\DELL\Desktop\DAA\p8\output> █

```