

Week 9:

- I. Given a graph, Design an algorithm and implement it using a program to implement Floyd-Warshall all pair shortest path algorithm.

Input Format:

The first line of input takes number of vertices in the graph.

Input will be the graph in the form of adjacency matrix or adjacency list. If a direct edge is not present between any pair of vertex (u,v), then this entry is shown as $\text{AdjM}[u,v] = \text{INF}$.

Output Format:

Output will be shortest distance matrix in the form of $V \times V$ matrix, where each entry (u,v) represents shortest distance between vertex u and vertex v.

Sample I/O Problem I:

Input: 5 0 10 5 5 INF INF 0 5 5 5 INF INF 0 INF 10 INF INF INF 0 20 INF INF INF 5 0	Output: Shortest Distance Matrix: 0 10 15 5 15 INF 0 5 5 5 INF INF 0 15 10 INF INF INF 0 20 INF INF INF 5 0
--------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------

- II. Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. You have to Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight w and has maximum value. You can take fractions of items, i.e. the items can be broken into smaller pieces so that you have to carry only a fraction x_i of item i, where $0 \leq x_i \leq 1$.

Input Format:

First input line will take number of items N which are provided.

Second input line will contain N space-separated array containing weights of all N items.

Third input line will contain N space-separated array containing values of all N items.

Last line of the input will take the maximum capacity w of knapsack.

Output Format:

First output line will give maximum value that can be achieved.

Next Line of output will give list of items selected along with their fraction of amount which has been taken.

Sample I/O Problem II:

Input: 6 6 10 3 5 1 3 6 2 1 8 3 5 16	Output: Maximum value : 22.33 item-weight 5-1 6-3 4-5 1-6 3-1
---------------------------------------------------------	-------------------------------------------------------------------------------------------

III. Given an array of elements. Assume $\text{arr}[i]$ represents the size of file i . Write an algorithm and a program to merge all these files into single file with minimum computation. For given two files A and B with sizes m and n , computation cost of merging them is $O(m+n)$. (Hint: use greedy approach)

Input Format:

First line will take the size n of the array.

Second line will take array s as an input.

Output Format:

Output will be the minimum computation cost required to merge all the elements of the array.

Sample I/O Problem III:

Input: 10 10 5 100 50 20 15 5 20 100 10	Output: 960
------------------------------------------------------	-----------------------

Solved example: Consider $\text{arr}[5] = \{ 10, 5, 100, 50, 20, 15 \}$. As per the brute force approach, first of all merge first two files (having 10 and 5 file size).

Cost of merging will be $= 10+5=15$.

List will become $\{15, 100, 50, 20, 15\}$.

Similarly, again merging first two files (i.e. having 15 and 100 file size).

Cost of merging will be $= 15+100=115$.

List will become $\{115, 50, 20, 15\}$.

For the subsequent steps the list becomes, $\{165, 20, 15\}$, $\{185, 15\}$ and $\{200\}$.

Therefore total cost of merging $= 15+115+165+185+200 = 680$.

But this is not minimum computation cost. To find minimum cost, consider the order $\text{arr}[5] = \{5, 10, 15, 20, 50, 100\}$. By applying the same approach, the total cost of merging $= 15+30+50+100+200 = 395$.