**Rajan Gautam**

**19BCP101**

**Div. II, CE 19**

**SOT, PDEU**

**Pandit Deendayal Energy University**

**School of Technology**

**Design & Analysis of Algorithm (20CP209P)**

**B. Tech - Computer Science & Engineering (Sem-IV)**

**Table of Contents**

[**Lab 5 Assignment: Solving Optimization Problems through Dynamic Programming 2**](#_Toc67953008)

[**AIM 1: To write a C/C++ Program (preferably) to implement the solution of Assemble Line Scheduling for given problem. Print the Line Number along with Station Number chosen for Optimal Solution. 2**](#_Toc67953009)

[**CODE: 2**](#_Toc67953010)

[**OUTPUT: 4**](#_Toc67953011)

[**AIM 2: Given a series of n arrays (of appropriate sizes) to multiply: A1×A2×⋯× An. Determine where to place parentheses to minimize the number of multiplications. 5**](#_Toc67953012)

[**CODE: 5**](#_Toc67953013)

[**OUTPUT: 8**](#_Toc67953014)

[**AIM 3: A thief enters a house for robbing it. He can carry a maximal weight of 60 kg into his bag. There are 4 items in the house with the following weights and values. 9**](#_Toc67953015)

[**CODE: 9**](#_Toc67953016)

[**OUTPUT: 11**](#_Toc67953017)

# Lab 5 Assignment: Solving Optimization Problems through Dynamic Programming

## AIM 1: To write a C/C++ Program (preferably) to implement the solution of Assemble Line Scheduling for given problem. Print the Line Number along with Station Number chosen for Optimal Solution.

### CODE:

1. **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Rajan Gautam \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**
2. **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 19BCP101 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**
4. **#include <stdio.h>**
6. **int min(int a, int b)**
7. **{**
8. **return (a < b) ? a : b;**
9. **}**
11. **int mySolver(int STATION, int a[][STATION], int t[][STATION], int \*e, int \*x)**
12. **{**
13. **int T1[STATION], T2[STATION], i, answer;**
15. **T1[0] = e[0] + a[0][0];**
16. **T2[0] = e[1] + a[1][0];**
18. **for (i = 1; i < STATION; ++i)**
19. **{**
20. **T1[i] = min(T1[i - 1] + a[0][i], T2[i - 1] + t[1][i] + a[0][i]);**
21. **T2[i] = min(T2[i - 1] + a[1][i], T1[i - 1] + t[0][i] + a[1][i]);**
22. **}**
24. **answer = min(T1[STATION - 1] + x[0], T2[STATION - 1] + x[1]);**
26. **for (int i = 0; i < STATION; i++)**
27. **{**
28. **if (T1[i] > T2[i])**
29. **{**
30. **printf("STATION => %d  LINE => %d \n", i, 2);**
31. **}**
32. **else**
33. **{**
34. **printf("STATION => %d  LINE => %d \n", i, 1);**
35. **}**
36. **}**
37. **return answer;**
38. **}**
40. **int main()**
41. **{**
42. **int ans, No\_of\_Station;**
44. **printf("Enter the Number of Station: ");**
45. **scanf("%d", &No\_of\_Station);**
47. **int a[2][No\_of\_Station], t[2][No\_of\_Station], e[2], x[2];**
49. **for (int i = 0; i < No\_of\_Station; i++)**
50. **{**
51. **scanf("%d", &a[0][i]);**
52. **}**
53. **for (int i = 0; i < No\_of\_Station; i++)**
54. **{**
55. **scanf("%d", &a[1][i]);**
56. **}**
57. **for (int i = 0; i < No\_of\_Station; i++)**
58. **{**
59. **scanf("%d", &t[0][i]);**
60. **}**
61. **for (int i = 0; i < No\_of\_Station; i++)**
62. **{**
63. **scanf("%d", &t[1][i]);**
64. **}**
65. **for (int i = 0; i < 2; i++)**
66. **{**
67. **scanf("%d", &e[i]);**
68. **}**
69. **for (int i = 0; i < 2; i++)**
70. **{**
71. **scanf("%d", &x[i]);**
72. **}**
74. **ans = mySolver(No\_of\_Station, a, t, e, x);**
76. **printf("%d", ans);**
78. **return 0;**
79. **}**

## OUTPUT:

## AIM 2: Given a series of n arrays (of appropriate sizes) to multiply: A1×A2×⋯× An. Determine where to place parentheses to minimize the number of multiplications.

**Array dimensions:**

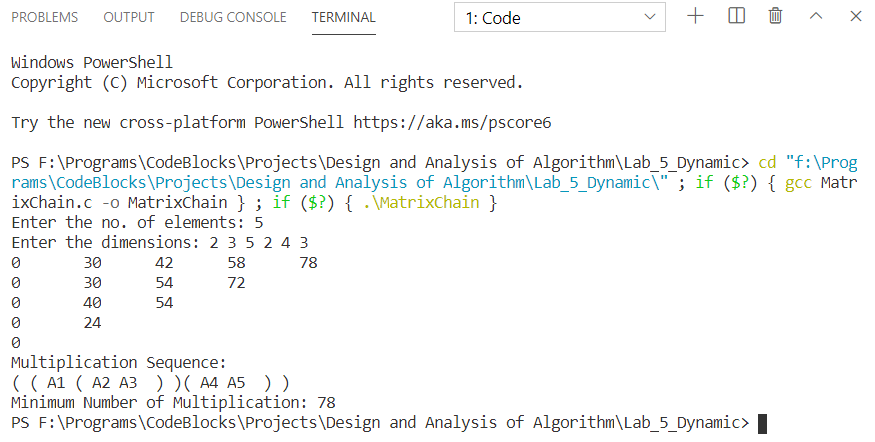
* **A1: 2 x 3**
* **A2: 3 x 5**
* **A3: 5 x 2**
* **A4: 2 x 4**
* **A5: 4 x 3**

**Print optimal locations for parentheses:**

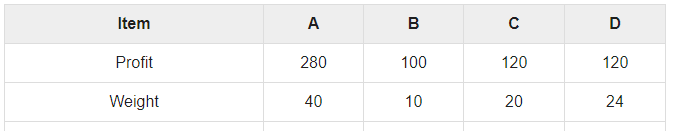
### CODE:

1. **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Rajan Gautam \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**
2. **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 19BCP101 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**
4. **#include <stdio.h>**
5. **#define INFINITY 999999;**
7. **int ss[6][6], mm[20][20], pp[20], i, j, n;**
9. **int min(int a, int b)**
10. **{**
11. **return (a < b) ? a : b;**
12. **}**
14. **int mySolver(int \*a, int s, int e)**
15. **{**
16. **if (s == e)**
17. **{**
18. **return 0;**
19. **}**
20. **int mini = \_\_INT\_MAX\_\_;**
22. **for (int i = s; i < e; i++)**
23. **{**
24. **mini = min(mini, mySolver(a, s, i) + mySolver(a, i + 1, e) + a[s - 1] \* a[i] \* a[e]);**
25. **}**
26. **return mini;**
27. **}**
29. **void MatrixMultiply(void)**
30. **{**
31. **long int q;**
32. **int i, j, k;**
33. **for (i = n; i > 0; i--)**
34. **{**
35. **for (j = i; j <= n; j++)**
36. **{**
37. **if (i == j)**
38. **mm[i][j] = 0;**
39. **else**
40. **{**
41. **for (k = i; k < j; k++)**
42. **{**
43. **q = mm[i][k] + mm[k + 1][j] + pp[i - 1] \* pp[k] \* pp[j];**
44. **if (q < mm[i][j])**
45. **{**
46. **mm[i][j] = q;**
47. **ss[i][j] = k;**
48. **}**
49. **}**
50. **}**
51. **}**
52. **}**
53. **}**
55. **void print\_optimal(int i, int j)**
56. **{**
57. **if (i == j)**
58. **printf("A%d ", i);**
59. **else**
60. **{**
61. **printf("( ");**
62. **print\_optimal(i, ss[i][j]);**
63. **print\_optimal(ss[i][j] + 1, j);**
64. **printf(" )");**
65. **}**
66. **}**
68. **int main()**
69. **{**
70. **printf("Enter the no. of elements: ");**
71. **scanf("%d", &n);**
72. **for (int i = 1; i <= n; i++)**
73. **{**
74. **for (int j = i + 1; j <= n; j++)**
75. **{**
76. **mm[i][i] = 0;**
77. **mm[i][j] = INFINITY;**
78. **ss[i][j] = 0;**
79. **}**
80. **}**
81. **printf("Enter the dimensions: ");**
82. **for (int i = 0; i <= n; i++)**
83. **{**
84. **scanf("%d", &pp[i]);**
85. **}**
87. **MatrixMultiply();**
89. **for (i = 1; i <= n; i++)**
90. **{**
91. **for (j = i; j <= n; j++)**
92. **{**
93. **printf("%d \t", mm[i][j]);**
94. **}**
95. **printf("\n");**
96. **}**
98. **i = 1;**
99. **j = n;**
100. **printf("Multiplication Sequence: \n");**
102. **print\_optimal(i, j);**
104. **printf("\nMinimum Number of Multiplication: %d", mySolver(pp, 1, n));**
106. **return 0;**
107. **}**

### OUTPUT:



## AIM 3: A thief enters a house for robbing it. He can carry a maximal weight of 60 kg into his bag. There are 4 items in the house with the following weights and values.



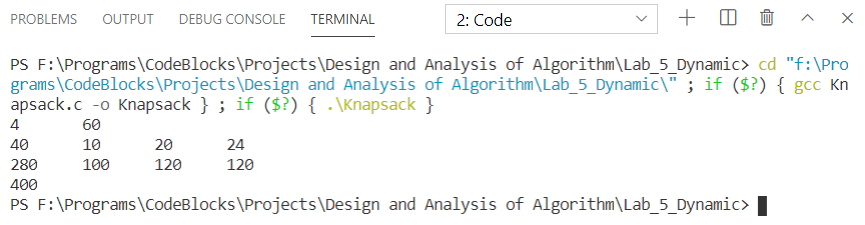
**1. Perform the 0-1 Knapsack and print the maximum weight and value of Knapsack using Dynamic programming.**

**2. Compare your answers with greedy approach.**

### CODE:

1. **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Rajan Gautam \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**
2. **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 19BCP101 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**
4. **#include <stdio.h>**
6. **int max(int a, int b)**
7. **{**
8. **return (a > b) ? a : b;**
9. **}**
11. **int myKnapsack(int w, int wt[], int val[], int n)**
12. **{**
13. **if (n == 0 || w == 0)**
14. **{**
15. **return 0;**
16. **}**
17. **if (wt[n - 1] > w)**
18. **{**
19. **return myKnapsack(w, wt, val, n - 1);**
20. **}**
21. **else**
22. **{**
23. **return max(val[n - 1] + myKnapsack(w - wt[n - 1], wt, val, n - 1), myKnapsack(w, wt, val, n - 1));**
24. **}**
25. **}**
27. **void solve()**
28. **{**
29. **int n, w, z;**
30. **scanf("%d %d", &n, &w);**
31. **int wt[n], val[n];**
32. **for (int i = 0; i < n; i++)**
33. **{**
34. **scanf("%d", &wt[i]);**
35. **}**
36. **for (int i = 0; i < n; i++)**
37. **{**
38. **scanf("%d", &val[i]);**
39. **}**
40. **z = myKnapsack(w, wt, val, n);**
41. **printf("%d", z);**
42. **}**
44. **int main()**
45. **{**
46. **solve();**
47. **return 0;**
48. **}**

### OUTPUT:



#### Comparison with Greedy Knapsack (Fractional Knapsack):

Here, we cannot divide the items in parts to obtain maximum profit. So, the output will be different.

For the given input the dynamic programming approach will allow us to take the 1st and 3rd item.

* **Profit = 280+120=400 & weight = 40+20 = 60**

While the greedy approach will give 440 as output (for profit take maximum value/kg items so take the 1st and 3rd item.

* **Profit = 7\*40+10\*10+6\*10 = 440 and weigh t= 40+10+10 = 60**

Because of both approaches are different they will give different output. The greedy one will try to divide the items while dynamic programming will check overall and gives the best possible solution without dividing the items

**Link:** <https://github.com/rgautam320/Design-and-Analysis-of-Algorithm-Lab/tree/master/Lab_5_Dynamic>