Rajan Gautam
19BCP101

Div. II, CE 19

SOT, PDEU

Pandit Deendayal Energy University

School of Technology

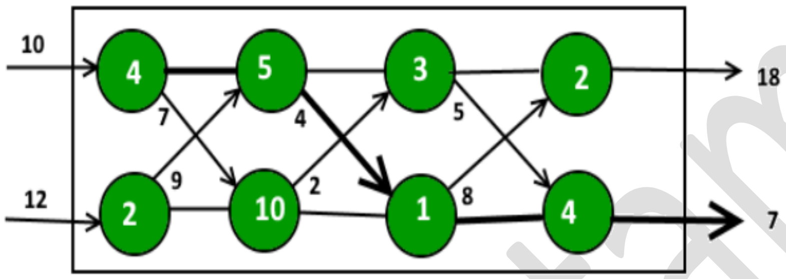
Design & Analysis of Algorithm (20CP209P)

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

Table of Contents

Lab 5 Assignment: Solving Optimization Problems through Dynamic Programming

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.

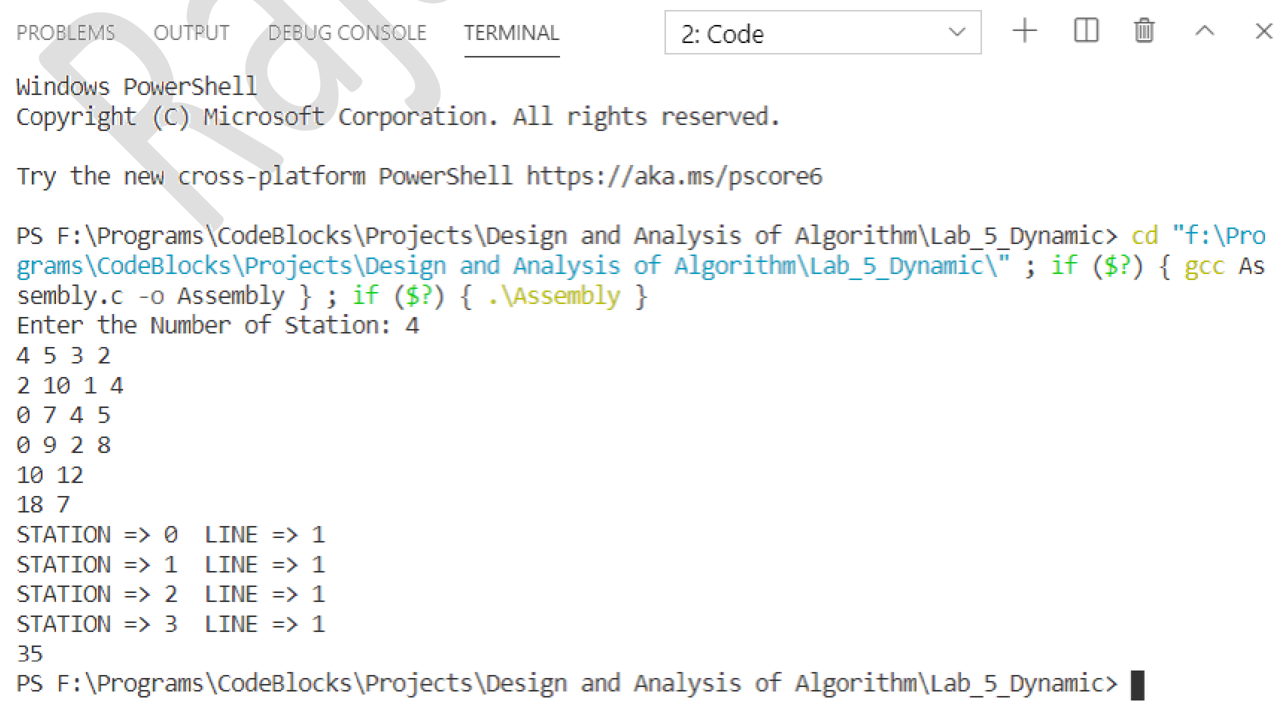


CODE:

```
/*********************** Rajan Gautam ***********************/
/************************** 19BCP101 *************************/

#include <stdio.h>

int min(int a, int b)
{
    return (a < b) ? a : b;
}

int mySolver(int STATION, int a[][STATION], int t[][STATION], int *e, int *x)
{
    int T1[STATION], T2[STATION], i, answer;

    T1[0] = e[0] + a[0][0];
    T2[0] = e[1] + a[1][0];

    for (i = 1; i < STATION; ++i)
    {
        T1[i] = min(T1[i - 1] + a[0][i], T2[i - 1] + t[1][i] + a[0][i]);
```

```c
        T2[i] = min(T2[i - 1] + a[1][i], T1[i - 1] + t[0][i] + a[1][i]);
    }

    answer = min(T1[STATION - 1] + x[0], T2[STATION - 1] + x[1]);

    for (int i = 0; i < STATION; i++)
    {
        if (T1[i] > T2[i])
        {
            printf("STATION => %d  LINE => %d \n", i, 2);
        }
        else
        {
            printf("STATION => %d  LINE => %d \n", i, 1);
        }
    }
    return answer;
}

int main()
{
    int ans, No_of_Station;

    printf("Enter the Number of Station: ");
    scanf("%d", &No_of_Station);

    int a[2][No_of_Station], t[2][No_of_Station], e[2], x[2];

    for (int i = 0; i < No_of_Station; i++)
    {
        scanf("%d", &a[0][i]);
    }
    for (int i = 0; i < No_of_Station; i++)
    {
        scanf("%d", &a[1][i]);
    }
    for (int i = 0; i < No_of_Station; i++)
    {
        scanf("%d", &t[0][i]);
    }
    for (int i = 0; i < No_of_Station; i++)
    {
        scanf("%d", &t[1][i]);
    }
    for (int i = 0; i < 2; i++)
    {
        scanf("%d", &e[i]);
    }
    for (int i = 0; i < 2; i++)
    {
        scanf("%d", &x[i]);
    }

    ans = mySolver(No_of_Station, a, t, e, x);

    printf("%d", ans);

    return 0;
}
```

OUTPUT:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS F:\Programs\CodeBlocks\Projects\Design and Analysis of Algorithm\Lab_5_Dynamic> cd "f:\Pro
grams\CodeBlocks\Projects\Design and Analysis of Algorithm\Lab_5_Dynamic\" ; if ($?) { gcc As
sembly.c -o Assembly } ; if ($?) { .\Assembly }
Enter the Number of Station: 4
4 5 3 2
2 10 1 4
0 7 4 5
0 9 2 8
10 12
18 7
STATION => 0  LINE => 1
STATION => 1  LINE => 1
STATION => 2  LINE => 1
STATION => 3  LINE => 1
35
PS F:\Programs\CodeBlocks\Projects\Design and Analysis of Algorithm\Lab_5_Dynamic>
```

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**

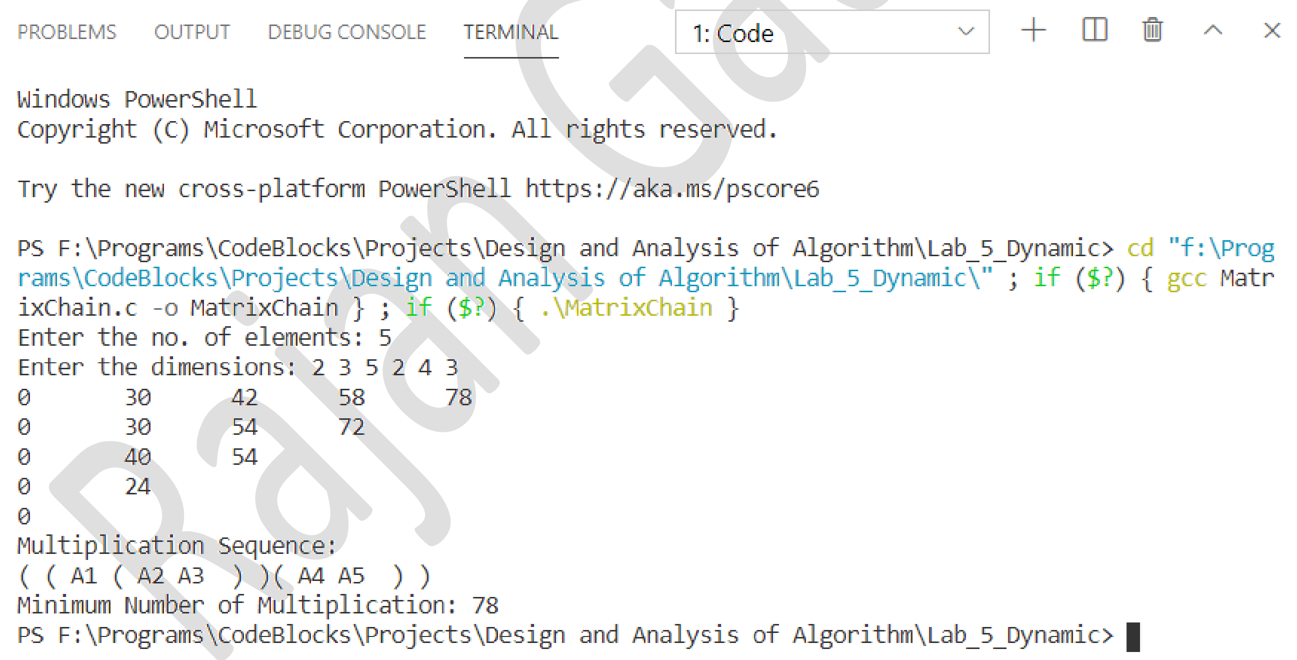
Print optimal locations for parentheses:

CODE:

```
/*********************** Rajan Gautam ***********************/
/************************* 19BCP101 *************************/

#include <stdio.h>
#define INFINITY 999999;

int ss[6][6], mm[20][20], pp[20], i, j, n;

int min(int a, int b)
 {
     return (a < b) ? a : b;
 }

 int mySolver(int *a, int s, int e)
 {
     if (s == e)
     {
         return 0;
     }
     int mini = __INT_MAX__;

     for (int i = s; i < e; i++)
     {
         mini = min(mini, mySolver(a, s, i) + mySolver(a, i + 1, e)
 + a[s - 1] * a[i] * a[e]);
```

```c
    }
    return mini;
}

void MatrixMultiply(void)
{
    long int q;
    int i, j, k;
    for (i = n; i > 0; i--)
    {
        for (j = i; j <= n; j++)
        {
            if (i == j)
                mm[i][j] = 0;
            else
            {
                for (k = i; k < j; k++)
                {
                    q = mm[i][k] + mm[k + 1][j] + pp[i - 1] * pp[k] * pp[j];
                    if (q < mm[i][j])
                    {
                        mm[i][j] = q;
                        ss[i][j] = k;
                    }
                }
            }
        }
    }
}

void print_optimal(int i, int j)
{
    if (i == j)
        printf("A%d ", i);
    else
    {
            printf("( ");
            print_optimal(i, ss[i][j]);
            print_optimal(ss[i][j] + 1, j);
            printf(" )");
        }
}

int main()
{
    printf("Enter the no. of elements: ");
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
    {
        for (int j = i + 1; j <= n; j++)
        {
            mm[i][i] = 0;
            mm[i][j] = INFINITY;
            ss[i][j] = 0;
        }
    }
    printf("Enter the dimensions: ");
    for (int i = 0; i <= n; i++)
    {
        scanf("%d", &pp[i]);
    }

    MatrixMultiply();

    for (i = 1; i <= n; i++)
    {
        for (j = i; j <= n; j++)
        {
            printf("%d \t", mm[i][j]);
        }
        printf("\n");
    }

    i = 1;
    j = n;
    printf("Multiplication Sequence: \n");

    print_optimal(i, j);

    printf("\nMinimum Number of Multiplication: %d", mySolver(pp,
  1, n));

    return 0;
}
```

OUTPUT:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS F:\Programs\CodeBlocks\Projects\Design and Analysis of Algorithm\Lab_5_Dynamic> cd "f:\Prog
rams\CodeBlocks\Projects\Design and Analysis of Algorithm\Lab_5_Dynamic\" ; if ($?) { gcc Matr
ixChain.c -o MatrixChain } ; if ($?) { .\MatrixChain }
Enter the no. of elements: 5
Enter the dimensions: 2 3 5 2 4 3
0       30      42      58      78
0       30      54      72
0       40      54
0       24
0
Multiplication Sequence:
( ( A1 ( A2 A3  ) )( A4 A5  ) )
Minimum Number of Multiplication: 78
PS F:\Programs\CodeBlocks\Projects\Design and Analysis of Algorithm\Lab_5_Dynamic>
```

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.

Item	A	B	C	D
Profit	280	100	120	120
Weight	40	10	20	24

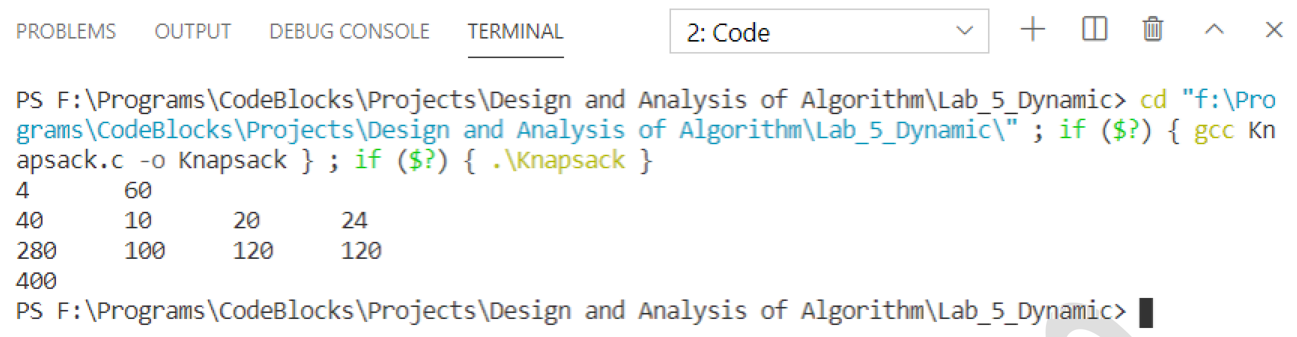
1. Perform the 0-1 Knapsack and print the maximum weight and value of Knapsack using Dynamic programming.

2. Compare your answers with greedy approach.

CODE:

```
/************************ Rajan Gautam ************************/
/************************** 19BCP101 *************************/

#include <stdio.h>

int max(int a, int b)
{
    return (a > b) ? a : b;
}

int myKnapsack(int w, int wt[], int val[], int n)
{
    if (n == 0 || w == 0)
    {
        return 0;
    }
    if (wt[n - 1] > w)
    {
        return myKnapsack(w, wt, val, n - 1);
    }
    else
    {
            return max(val[n - 1] + myKnapsack(w - wt[n - 1], wt, val, n - 1), myKnapsack(w, wt, val, n - 1));
        }
}

void solve()
{
    int n, w, z;
    scanf("%d %d", &n, &w);
    int wt[n], val[n];
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &wt[i]);
    }
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &val[i]);
    }
    z = myKnapsack(w, wt, val, n);
    printf("%d", z);
}

int main()
{
    solve();
    return 0;
}
```

OUTPUT:



```
PS F:\Programs\CodeBlocks\Projects\Design and Analysis of Algorithm\Lab_5_Dynamic> cd "f:\Pro
grams\CodeBlocks\Projects\Design and Analysis of Algorithm\Lab_5_Dynamic\" ; if ($?) { gcc Kn
apsack.c -o Knapsack } ; if ($?) { .\Knapsack }
4       60
40      10      20      24
280     100     120     120
400
PS F:\Programs\CodeBlocks\Projects\Design and Analysis of Algorithm\Lab_5_Dynamic>
```

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