CSE 2003- Data Structures and Algorithms

J-Component Report

A Project report titled-

Optimized path for Product manufacturing

By

19BCE1679 Sunrit Sarkar
19BCE1690 Shreya Agrawal
19BCE1698 Sam Methuselah

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGENEERING

Submitted to

Dr. R. Rajalakshmi

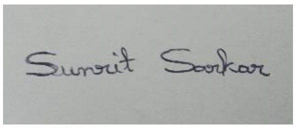
School of Computer Science and Engineering

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

November 2020

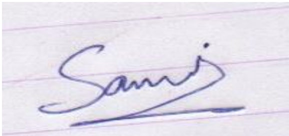
DECLARATION BY THE CANDIDATES

We, hereby declare that the report titled "**Optimized path for Product manufacturing"** submitted by us to VIT Chennai is a record of bona-fide work undertaken by us under the supervision of **Dr. R. Rajalakshmi, Associate Professor, SCOPE, Vellore Institute of Technology, Chennai.**



Signature of the Candidate 1: *Sunrit*

Signature of the Candidate 2: *Shreya*



Signature of the Candidate 3: *Sam*

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. R. Rajalakshmi,** School of Computer Science and Engineering for her consistent encouragement and valuable guidance offered to us throughout the course of the project work.

We are extremely grateful to **Dr. R. Jagadeesh Kannan, Dean,** School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Chennai, for extending the facilities of the School towards our project and for his unstinting support.

We express our thanks to our **Head of the Department** for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the courses.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

BONAFIDE CERTIFICATE

Certified that this project report entitled "**Optimized Path for Product Manufacturing"** is a bona-fide work of **Sunrit Sarkar (19BCE1679), Shreya Agrawal (19BCE1690) and Sam Methuselah (19BCE1698)** carried out the "J"-Project work under my supervision and guidance for **CSE2003-Data Structures and Algorithms.**

Dr. R. Rajalakshmi

SCOPE

TABLE OF CONTENTS

INTRODUCTION

Abstract

Assembly line scheduling helps us to understand and know how tasks are to be assigned to workstations, so that the predetermined goal is achieved. Minimization of the number of workstations and maximization of the production rate are the most common goals. This project thus addresses the problem of scheduling the cloth manufacturing unit's time on identical parallel machines (assembly lines) to minimize total weighted time required for manufacturing using Dynamic Programming.

Dynamic Programming uses a bottom up approach to build up the final solution. The solution of a problem is formulated recursively in terms of sub problems however we construct the solution for the bigger problem by first solving the smaller problems then combining the solutions of the sub problems and examining a model that incorporates the efficiency/timeliness conflict in practice. We propose properties of an optimal solution for the purpose of exposing to reduce the total time required for the production in the cloth production unit.

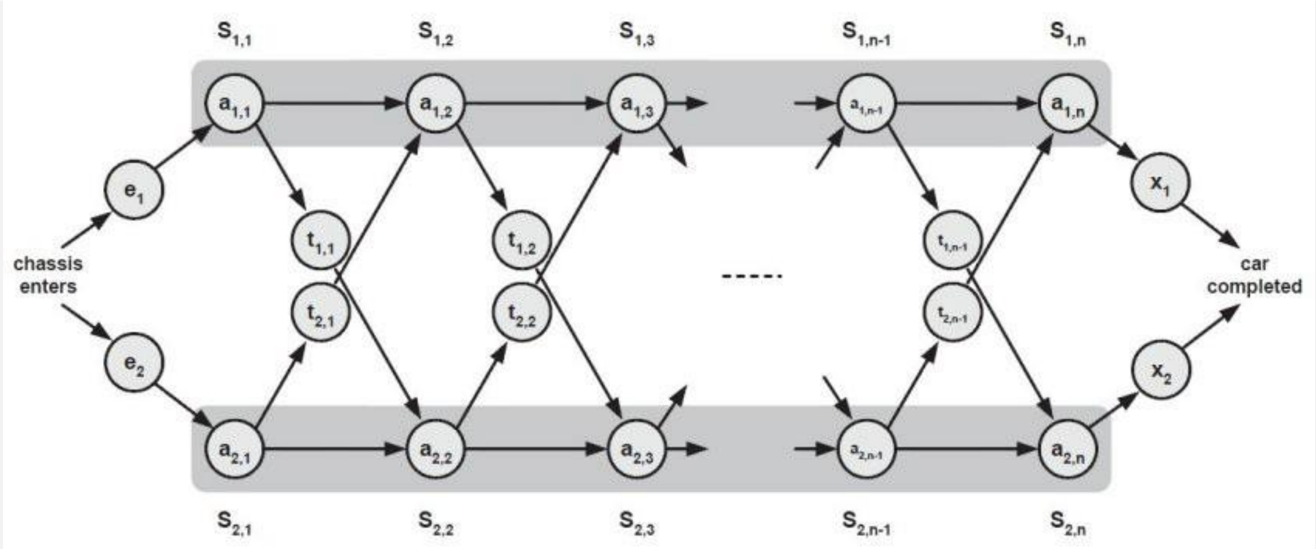
Problem Statement

A cloth manufacturing company has, for example two assembly lines, each with n stations. A station is denoted by $S_{i,j}$ where i denotes the assembly line the station is on and j denotes the number of the station. The time taken per station is denoted by $a_{i,j}$. Each station is dedicated to do some sort of work in the manufacturing process. So, a cloth must pass through each of the n stations in order before exiting the company. The parallel stations of the two assembly lines perform the same task. After it passes through station $S_{i,j}$, it will continue to station $S_{i,j+1}$ unless it decides to transfer to the other line. Continuing on the same line incurs no extra cost, but transferring from line i at station $j - 1$ to station j on the other line takes time $t_{i,j}$. Each assembly line takes an entry time e_i and exit time x.

ASSEMBLY LINE SCHEDULING

We know that in automobile industry, automobiles are produced using assembly lines. There are multiple lines that are working together to produce products. Then a finished auto exits at the end of the line.

The problem is that which line we should choose next from any station that will give best time utilization for company for one auto.



The main goal of assembly line scheduling is to give best route or can say fastest from all assembly line.

In above the diagram we have two main assembly line consider as LINE 1 and LINE 2.

- e[i]: entry time in assembly line i [here i=1,2]
- x[i]: exit time from assembly line i
- a[i,j]: Time required at station S[i,j] (assembly line i, stage j)[i=1 to n] because Every station has some dedicated job that needs to done.
- t[i,j]: Time required to transit from station S[i,j] to the other assembly line.

Normally, once a chassis enters an assembly line, it passes through that line only. The time to go from one station to the next within the same assembly line is negligible.

Occasionally, a special rush order comes in, and the customer wants the automobile to be manufactured as quickly as possible. For the rush orders, the chassis still passes through the n stations in order, but the factory manager may switch the partially-completed auto from one assembly line to the other after any station.

To get jobs done we will use Dynamic Programming.

Objective: To find the optimal scheduling i.e., the fastest way from start to exit.

***Note*: Let fi [j] denotes the fastest way from start to station S[i,j]** .

An optimal solution to a problem is determined using optimal solutions to sub problems (in turn, sub sub problems and so on).

The immediate question is, how to break the problem in to smaller sub problems?

The answer is: If we know the minimum time taken by the chassis to leave station S[i,j−1] then the minimum time taken to leave station S[i,j] can be calculated quickly by combining a[i,j] and t[i,j] .

Final Solution: f *optimal* = min{f1[n] + x1, f2[n] + x2}.

We can take f1[1] = e1 + a[1,1] and f2[1] = e2 + a[2,1].

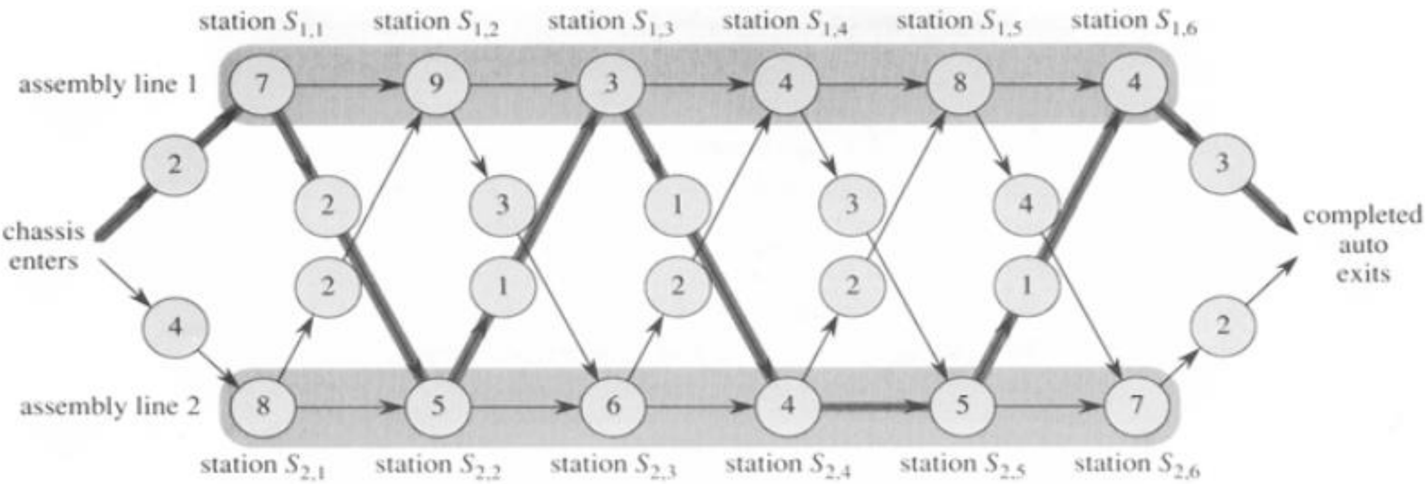
Recursive Solution: The chassis at station S[1,j] can come either from station S[1,j−1] or station S[2,j−1] (Since, the tasks done by S[1,j] and S[2,j] are same). But if the chassis comes from S[2,j−1], it additionally incurs the transfer cost to change the assembly line (like t[2,j-1]). Thus, the recursion to reach the station j in assembly line i are as follows:

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & \text{if } j = 1 \\ \min\{f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}\} & \text{if } j \geq 2 \end{cases}$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1} & \text{if } j = 1 \\ \min\{f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j}\} & \text{if } j \geq 2 \end{cases}$$

The following diagram shows the tables created to find the shortest path.
It includes:

- If one should continue in the same assembly line or
- If one should jump to another assembly line to reduce time used



(a)

j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$l^* = 1$

The algorithm for Assembly Line Scheduling with Dynamic Programming-

The algorithm: **Fastest-way**(a, t, e, x, n) Source: CLRS

where a denotes the assembly costs, t denotes the transfer costs, e denotes the entry costs, x denotes the exit costs and n denotes the number of assembly stages.

$f_1[1] = e_1 + a_{1,1}$
$f_2[1] = e_2 + a_{2,1}$
for $j = 2$ to n
  if $((f_1[j-1] + a_{1,j}) \leq (f_2[j-1] + t_{2,j-1} + a_{1,j}))$ then
    $f_1[j] = f_1[j-1] + a_{1,j}$ and $l_1[j] = 1$ /* l_p denotes the line p */
  else
    $f_1[j] = f_2[j-1] + t_{2,j-1} + a_{1,j}$ and $l_1[j] = 2$
  if $((f_2[j-1] + a_{2,j}) \leq (f_1[j-1] + t_{1,j-1} + a_{2,j}))$ then
    $f_2[j] = f_2[j-1] + a_{2,j}$ and $l_2[j] = 2$
  else
    $f_2[j] = f_1[j-1] + t_{1,j-1} + a_{2,j}$ and $l_2[j] = 1$
end for
if $(f_1[n] + x_1 \leq f_2[n] + x_2)$ then
  $f^{OPT} = f_1[n] + x_1$ and $l^{OPT} = 1$
else
  $f^{OPT} = f_2[n] + x_2$ and $l^{OPT} = 2$

Explanation of algorithm:

1. If, as discussed above, we consider 2 assembly line. Then f1[1] and f2[1] is defined as algorithm by adding starting cost and first station cost.
2. Then applies our recursive solution for n station points. ere l1[j] denotes from which assembly line chassis has come.
3. At the last **f*opt*** gives you final solution.

The **time complexity** of the above dynamic programming implementation of the assembly line scheduling problem is **O(n)**.

METHODOLOGY

A simple recursive technique could be used to calculate that the brute force method makes the time complexity exponential, that is, it becomes **O(2^n).**

Thus, we strive to use the Dynamic Algorithm: Assembly Scheduling technique, which reduces the time complexity drastically and makes it **O(n).**

Step 1: Start

Step 2: (i) The fastest way to, say, a station S1,j (which signifies the j^{th} station in the first line) would be the least time it takes to complete the task in the S1,j-1 station and then directly going to the said station without any time delay.

(ii) As for the other alternative, the fastest way to S1,j can also be the least time taken in the previous station in line 2, that is, in the station S2,j-1 and then losing some amount of time and get transferred to line 1, and reach the said station.

Step 3: Symmetrical, same concept will be applied for the stations in the second line and the third line, say, S2,j, S3,j in order to calculate the fastest route to the said stations.

Step 4: Now, as the problem statement states, we have been given the entry time and the exit time of each of the lines, along with the manufacturing time each station take, we take n arrays (where, n is the number of lines) along with an array which stores the line number from which the corresponding station is taken for the optimized route.

Step 5: Store the time to reach the first station in of each line in each of the n-arrays respectively.

Step 6: Now calculate the time required to reach the next station in the particular line (say, Line k) from all the previous stations of all the n lines. The minimum of the calculated time to reach that station is stored in the corresponding array.

Step 7: Store the line number of the previous station from which the minimum time was taken to reach the present station, in the corresponding path array of that line, that is, $(n+k)^{th}$ array (say, array lk[ ])

Step 8: Step 6 is repeated for the stations of same level for each line (Station 2 of Line 1, Line 2, Line 3 and so on).

Step 9: Repeat Step 6 and Step 7 and Step 8 till the last station is reached in each of the lines.

Step 10: After calculating the minimum time required to reach the last station of each of the lines, we add the exit time of the corresponding lines to each of them.

Step 11: We calculate the minimum time from the list from Step 10 and this line number is the exit for the optimized product manufacturing (say, Line r)

Step 12: As for our output, the minimum time which we calculated in Step 11 is the most optimized time required for the complete manufacturing of the product.

Step 13: Now, we start backtracking from the $(n+r)^{th}$ array (say, lr[ ]). This is done by going to that particular path array whose number is present as an element in the present array. For example, we exited from line 2 (r), and the element present in l2[n-1] (where, n is the number of station) is 1, so that means that the previous station from line 1 was used (we print the station number along with the line number) , and then the element present in l1[n-2] is 3, which signifies that the station from line 3 was used.

Step 14: After we finish backtracking, we would have had printed the most optimized path which the product would take for its most efficient completion, in the reverse order.

Step 15: Stop.

IMPLEMENTATION

Video Link- https://youtu.be/bG1WqwpPW14

Let the input be in the form:

(i) There are m stations and n lines.
(ii) The entry time is given by ei for each line.
(iii) The exit time is given by xi for each line.
(iv) The time taken in station ‘j’ on line ‘i’ is ai,j.
(v) The time taken to transfer the product from one station from a line to the next station in a different line is given by ti,j

The way we are going to implement our solution to the problem statement is by this method: if we know the minimum time taken by the product to leave station Si,j-1 then the minimum time taken to leave station Si,j can be calculated with less complexity by combining ai,j and ti,j .

Final Solution:
F= min{f1[m] + x1, f2[m] + x2, f3[m] + x3, }

Base Cases:
f1[1] = e1 + a1,1 and f2[1] = e2+a2,1 and f3[1] = e3+a3,1.

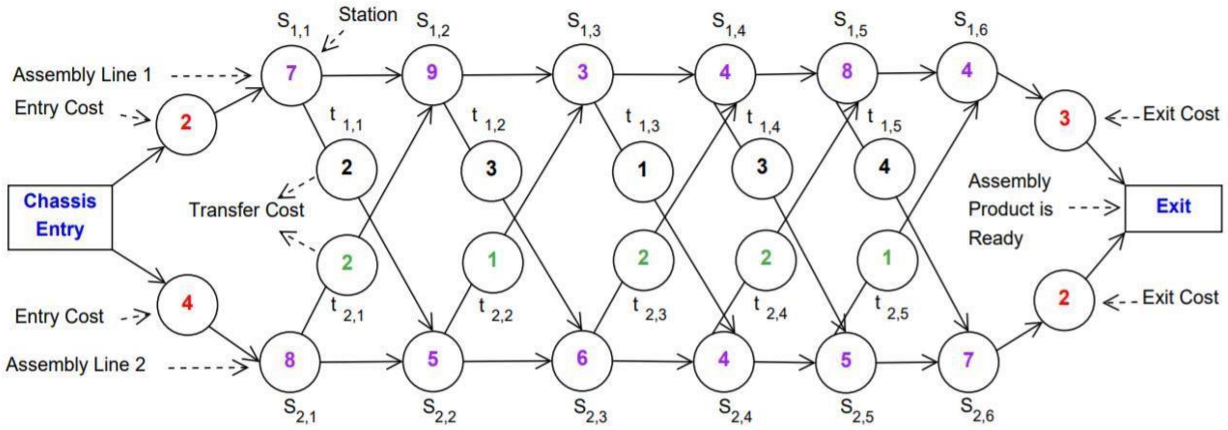
The recursive technique (using Assembly Line Scheduling) to reach a particular station ‘j’ in the assembly line ‘i’ can be given by the function:

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & \text{if } j = 1 \\ \min\{f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}\} & \text{if } j \geq 2 \end{cases}$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1} & \text{if } j = 1 \\ \min\{f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j}\} & \text{if } j \geq 2 \end{cases}$$

Here, we are assuming that the value of n is 2, that is, the number of assembly lines are 2. Hence, the functions give the conditions for two if its assembly lines.

Pictorial Representation of the implementation of the solution:



Here, the number of stations are 6 (m) and the lines are 2 (n).

As stated in the algorithm, we will be having 2n arrays which will be storing our data. The first n arrays store the minimum time required to reach a particular station in that particular line, in the location corresponding to the station number. The next n array is to store the line number of which the previous station was part of, from which we came to the current station with the shortest path.

From our algorithm we calculate the elements of the 2n arrays (in this case 4 arrays):

f1= [9, 18, 20, 24 32, 35]   l1= [1, 2, 1, 1, 2]

f2= [12, 16, 22, 25, 30, 37]   l2= [1, 2, 1, 2, 2]

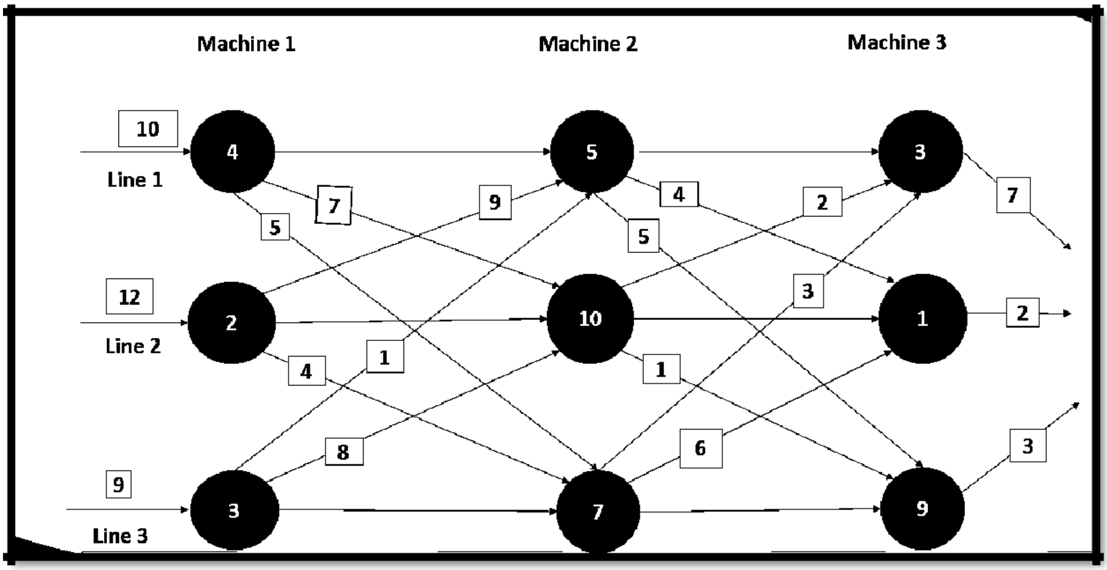
Note that li[j] stores the line number that minimizes the total cost from start to station (j – 1). The optimal solution from start to exit comes from the station S1,6. The fastest way to reach the station S1,6 from start, f1[6], comes from f2[5] i.e., from the station S2,5. The fastest way to reach the station S2,5 from start, f2[5], comes from f2[4] i.e., from the station S2,4. The fastest way to reach the station S2,4 from start, f2[4], comes from f1[3] i.e., from the station S1,3.The fastest way to reach the station S1,3 from start, f1[3], comes from f2[2] i.e., from the station S2,2.The fastest way to reach the station S2,2 from start, f2[2], comes from f1[1] i.e., from the station S1,1, which is one of the base cases.

Thus, the optimal schedule is:

min(f1[6] + x1, f2[6] + x) = min (35+3, 37+2) = 38 (i.e. from Line 1)

Start → e1 → S1,1 → t1,1 → S2,2 → t2,2 → S1,3 → t1,3 → S2,4 → S2,5 → t2,5 → S1,6 → x1 → exit

Let's take an Example (n=3):



The arrays used in the code for the implementation of the above Assembly Line have the following values:

w1[][]

4	**5**	**3**
2	**10**	**1**
3	**7**	**9**

time[][]

14	**18**	**21**	**28**
14	**24**	**23**	**25**
12	**19**	**28**	**31**

path[][]

1	**3**	**1**
2	**2**	**1**
3	**3**	**3**

Note:

- w2 is a three-dimensional array which represents the time taken by the chassis to travel between machines.
- time[i][j]= It is the minimum time taken to cross the (j+1)th machine of the (i+1)th line.
- path[i][j]=It is the line number of the previous machine ((j+1)th) machine.

CODE-

```
#include<stdio.h>

#include<stdlib.h>


int lines;

int machines;

int check, check1=1;

int oppath[100]; //optimized path

int time[100][100];


int flag=0;


void display(int e1[], int e2[], int **w1, int ***w2)

{

	int i,j,k;

	printf("1. The entry time for each of the line:\n\n");

	for (i=0; i<lines; i++)

	{

		printf("Line %d: ", i+1);

		printf("%d", e1[i]);

		printf("\n");

	}

	printf("\n2. The time taken to process the product in each machine:\n\n");

	for(i=0; i<lines; i++)

	{

		printf("Line %d\n", i+1);

		for(j=0; j<machines; j++)
```

```
        {
            printf("Machine %d: ", j+1);
            printf("%d", w1[i][j]);
            printf("\n");
        }
    }
    printf("\n3. The time taken to travel from a machine in one level to the machines in the other level:\n\n");
    for(i=0; i<lines; i++)
    {
        for(j=0; j<machines-1; j++)
        {
            printf("The time taken to travel from Machine %d of Line %d to: \n", j+1, i+1);
            for(k=0; k<lines; k++)
            {
                printf("Machine %d of line %d: ", j+2, k+1);
                printf("%d", w2[i][j][k]);
                printf("\n");
            }
        }
    }
    printf("\n4. The exit time for each of the line: \n\n");
    for(i=0; i<lines; i++)
    {
        printf("Line %d: ", i+1);
        printf("%d", e2[i]);
        printf("\n");
```

```
	}
	printf("\n");
	printf("Press any key to Continue...");
	getch();
	system("cls");
}


void process(int e1[], int e2[], int **w1, int ***w2)
{
	int path[lines][machines-1];
	int i,j,k,t1,minpath,tmin;
	char x;
	for(i=0; i<lines; i++)
	{
		time[i][0]=e1[i]+w1[i][0];
	}
	for(i=0; i<machines; i++)
	{
		for(j=0; j<lines; j++)
		{
			tmin=time[0][i]+w1[j][i+1]+w2[0][i][j];
			minpath=1;
			for(k=0; k<lines; k++)
			{
				if(tmin>time[k][i]+w1[j][i+1]+w2[k][i][j])
				{
```

```
                    tmin=time[k][i]+w1[j][i+1]+w2[k][i][j];
                    minpath=k-1;
                }
            }
            time[j][i+1]=tmin;
            path[j][i]=minpath;
        }
}

for(i=0; i<lines; i++)
{
        time[i][machines]=time[i][machines-1]+e2[i];
}
int min=time[0][machines];
int bestendline=1;
for(i=1; i<lines; i++)
{
        if(min>time[i][machines])
        {
            min=time[i][machines];
            bestendline=i+1;
        }
}

if(check1!=0)
{
        printf("The least time required for a product to be made is : %d\n\n", time[bestendline-
```

```
1][machines]);
                check1=1;
        }

        oppath[machines-1]=bestendline;
        int temp=oppath[machines-1]-1;
        for(i=machines-2; i>=0; i--)
        {
                oppath[i]=path[temp][i];
                temp=oppath[i]-1;
        }

        if(check1!=0)
        {
                for(i=0; i<machines; i++)
                {
                        printf("%d.) Machine %d of Line %d\n", i+1, i+1, oppath[i]);
                }
                printf("\n\n");
                printf("Press any key to return to the HOME page... ");
                getch();
                system("cls");
                check1=1;
        }
}

void display1(int e1[], int e2[], int **w1, int ***w2)
```

```
{
	check1=0;
	process(e1,e2,w1,w2);
	check1=1;
	int n,i;
	printf("Enter the machine number for which the optimized path and time is to be
displayed:\n");
	scanf("%d", &n);
	if(n>machines)
	{
		printf("The number exceeds the total number of machines.\n");
		printf("Displaying the output for n=total number of machines!\n");
		n=machines;
	}
	printf("The least time required to reach and operate in machine %d is:", n);
	int bestendline;
	bestendline=oppath[n-1];
	printf("%d\n", time[bestendline-1][n-1]);
	for(i=0; i<n; i++)
	{
		printf("%d.) Machine %d of Line %d\n", i+1, i+1, oppath[i]);
	}
	printf("\n");
	printf("Press any key to Continue...");
	getch();
	system("cls");
}
```

```
int inputchoice(int count, int ch1)
{
	int ch;
	char x;
	if(count<4)
	{
		printf("\nWhat do you want to input?? \n");
		printf("1. The time taken to process the product in each machine \n");
		printf("2. The entry time for each of the line \n");
		printf("3. The exit time for each of the line \n");
		printf("4. The time taken to travel from a machine in one level to the machine in the other level\n");
		printf("5. To reset all the inputs\n");
		printf("6. To start over\n\n");
		printf("Enter the choice: ");
		scanf("%d", &ch);
		printf("\n");
		return ch;
	}
	else
	{
		printf("You have entered data successfully...");
		printf("\n\n");
		printf("Double ENTER to calculate the optimised path... ");
		getch();
		scanf("%c", &x);
```

```
        system("cls");
        return 5;
    }
}

void input(int e1[], int e2[], int **w1, int ***w2)
{
    int flag=0, flag1=0, flag2=0, flag3=0;
    char x;
    int i,j,k;
    int ch=0;
    int count=0;
    ch=inputchoice(count,ch);
    while(count<4)
    {
        switch(ch)
        {
            case 1:
                if(flag==1)
                {
                    printf("You have already input this data!\n");
                }
                else
                {
                    flag=1;
                    count=count+1;
```

```
printf("=======================================================================================================================\n\n");
            for(i=0; i<lines; i++)
            {
                printf("Lines %d\n", i+1);
                for(j=0; j<machines; j++)
                {
                    printf("Machine %d: ", j+1);
                    scanf("%d", &w1[i][j]);
                    printf("\n");
                }
            }
        }
        printf("\n");
        printf("Press any key to Continue...");
        getch();
        scanf("%c", &x);
        system("cls");
        ch=inputchoice(count,ch);
        break;

    case 2:
        if(flag1==1)
        {
            printf("You have already input this data!\n");
        }
        else
```

```
                {
                        flag1=1;
                        count=count+1;

        printf("================================================================
===========================================================\n\n");
                        for(i=0; i<lines; i++)
                        {
                                printf("Line %d:", i+1);
                                scanf("%d", &e1[i]);
                                printf("\n\n");
                        }
                }
                printf("\n");
                printf("Press any key to Continue...");
                getch();
                scanf("%c", &x);
                system("cls");
                ch=inputchoice(count,ch);
                break;
        case 3:
                if(flag2==1)
                {
                        printf("You have already input this data!\n");
                }
                else
                {
                        flag2=1;
```

```
                        count=count+1;

    printf("===================================================================
===============================================================\n\n");
                        for(i=0; i<lines; i++)
                        {
                            printf("Line %d:", i+1);
                            scanf("%d", &e2[i]);
                            printf("\n\n");
                        }
                    }
                    printf("\n");
                    printf("Press any key to Continue...");
                    getch();
                    scanf("%c", &x);
                    system("cls");
                    ch=inputchoice(count,ch);
                    break;
            case 4:
                    if(flag3==1)
                    {
                        printf("You have already input this data!\n");
                    }
                    else
                    {
                        flag3=1;
                        count=count+1;
```

```
printf("==============================================================================================================================\n\n");
                for(i=0; i<lines; i++)
                {
                    for(j=0; j<machines-1; j++)
                    {
                        printf("The time taken to travel from Machine %d of Line %d to: \n", j+1, i+1);
                        for(k=0; k<lines; k++)
                        {
                            printf("Machine %d of line %d: \n", j+2, k+1);
                            scanf("%d", &w2[i][j][k]);
                        }
                    }
                }
            }
            printf("\n");
            printf("Press any key to Continue...");
            getch();
            scanf("%c", &x);
            system("cls");
            ch=inputchoice(count,ch);
            break;
        case 5:
            flag=0;
            flag1=0;
            flag2=0;
```

```
}
                    flag3=0;
                    count=0;
                    check=0;
                    printf("\n");
                    printf("The values haven been reset!\n\n");
                    printf("Press any key to Continue...");
                    getch();
                    scanf("%c", &x);
                    system("cls");
                    ch=inputchoice(count,ch);
                    break;
          case 6:
                    system("cls");
                    count=0;
                    check=0;
                    main();
                    break;
          default:
                    printf("Invalid input\n");
                    printf("Press any key to Try again...");
                    getch();
                    scanf("%c", &x);
                    system("cls");
                    ch=inputchoice(count,ch);
                    break;
          }
     }
```

```
    check=count;
}

int main()
{
    int c=0, ch;
    printf("\t\t\tHere we go!!\n\n");
    char x;
    printf("\tFirst enter the number of lines and machines in your factory...\n");
    printf("\tThe number of lines: ");
    scanf("%d", &lines);
    printf("\tThe number of machines in each line:");
    scanf("%d", &machines);
    printf("\n\n");
    printf("Press any key to Continue... ");
    getch();
    scanf("%c", &x);
    system("cls");
    int i,j,k;
    int e1[lines];
    int e2[lines];

    int *w1[lines];
    for (i=0; i<lines; i++)
    {
        w1[i]= (int *)malloc(machines* sizeof(int));
    }
```

```
int ***w2= (int***)malloc(lines *sizeof(int**));
for(i=0; i<lines; i++)
{
    w2[i]=(int**)malloc(machines*sizeof(int*));
    for(j=0; j<machines; j++)
    {
        w2[i][j]=(int*)malloc(machines* sizeof(int));
    }
}
again:
printf("                    HOME PAGE:\n");
printf("\nPress 1 to Input the data");
printf("\t\t\t\t(PLEASE INPUT THE DATA FIRST!!! IF DONE, IGNORE THIS MESSAGE)");
printf("\nPress 2 to Input Menu");
printf("\nPress 3 to Display the input data");
printf("\nPress 4 to Process and Display the most optimised path");
printf("\nPress 5 to END the program");
printf("\n\nEnter the choice:");
scanf("%d", &ch);
system("cls");
switch(ch)
{
    case 1:
        input(e1,e2,w1,w2);
        process(e1,e2,w1,w2);
```

```
                goto again;
                break;
            case 2:
                input(e1,e2,w1,w2);
                goto again;
                break;
            case 3:
                display(e1,e2,w1,w2);
                goto again;
                break;
            case 4:
                process(e1,e2,w1,w2);
                goto again;
                break;
            case 5:
                exit(0);
                break;
            default:
                printf("\nWRONG CHOICE!!!");
                break;
        }
        return 0;
}
```

SNAPSHOTS (Code)-

DSA.c

```c
#include<stdio.h>
#include<stdlib.h>

int lines;
int machines;
int check, check1=1;
int oppath[100]; //optimized path
int time[100][100];

int flag=0;

void display(int e1[], int e2[], int **w1, int ***w2)
{
    int i,j,k;
    printf("1. The entry time for each of the line:\n\n");
    for (i=0; i<lines; i++)
    {
        printf("Line %d: ", i+1);
        printf("%d", e1[i]);
        printf("\n");
    }
    printf("\n2. The time taken to process the product in each machine:\n\n");
    for(i=0; i<lines; i++)
    {
        printf("Line %d\n", i+1);
        for(j=0; j<machines; j++)
        {
            printf("    Machine %d: ", j+1);
            printf("%d", w1[i][j]);
            printf("\n");
        }
    }
    printf("\n3. The time taken to travel from a machine in one level to the machines in the other level:\n\n");
    for(i=0; i<lines; i++)
    {
        for(j=0; j<machines-1; j++)
```

DSA.c

```c
        {
            printf("The time taken to travel from Machine %d of Line %d to: \n", j+1, i+1);
            for(k=0; k<lines; k++)
            {
                printf("Machine %d of line %d: ", j+2, k+1);
                printf("%d", w2[i][j][k]);
                printf("\n");
            }
        }
    }
    printf("\n4. The exit time for each of the line: \n\n");
    for(i=0; i<lines; i++)
    {
        printf("Line %d: ", i+1);
        printf("%d", e2[i]);
        printf("\n");
    }
    printf("\n");
    printf("Press any key to Continue...");
    getch();
    system("cls");
}


void process(int e1[], int e2[], int **w1, int ***w2)
{
    int path[lines][machines-1];
    int i,j,k,t1,minpath,tmin;
    char x;
    for(i=0; i<lines; i++)
    {
        time[i][0]=e1[i]+w1[i][0];
    }
    for(i=0; i<machines; i++)
    {
        for(j=0; j<lines; j++)
```

DSA.c

```
        {
            tmin=time[0][i]+w1[j][i+1]+w2[0][i][j];
            minpath=1;
            for(k=0; k<lines; k++)
            {
                if(tmin>time[k][i]+w1[j][i+1]+w2[k][i][j])
                {
                    tmin=time[k][i]+w1[j][i+1]+w2[k][i][j];
                    minpath=k-1;
                }
            }
            time[j][i+1]=tmin;
            path[j][i]=minpath;
        }
    }

    for(i=0; i<lines; i++)
    {
        time[i][machines]=time[i][machines-1]+e2[i];
    }
    int min=time[0][machines];
    int bestendline=1;
    for(i=1; i<lines; i++)
    {
        if(min>time[i][machines])
        {
            min=time[i][machines];
            bestendline=i+1;
        }
    }

    if(check1!=0)
    {
        printf("The least time required for a product to be made is : %d\n\n", time[bestendline-1][machines]);
        check1=1;
    }
```

DSA.c

```

    oppath[machines-1]=bestendline;
    int temp=oppath[machines-1]-1;
    for(i=machines-2; i>=0; i--)
    {
        oppath[i]=path[temp][i];
        temp=oppath[i]-1;
    }

    if(check1!=0)
    {
        for(i=0; i<machines; i++)
        {
            printf("%d.) Machine %d of Line %d\n", i+1, i+1, oppath[i]);
        }
        printf("\n\n");
        printf("Press any key to return to the HOME page... ");
        getch();
        system("cls");
        check1=1;
    }
}

void display1(int e1[], int e2[], int **w1, int ***w2)
{
    check1=0;
    process(e1,e2,w1,w2);
    check1=1;
    int n,i;
    printf("Enter the machine number for which the optimized path and time is to be displayed:\n");
    scanf("%d", &n);
    if(n>machines)
    {
        printf("The number exceeds the total number of machines.\n");
        printf("Displaying the output for n=total number of machines!\n");
        n=machines;
```

DSA.c

```c
    }
    printf("The least time required to reach and operate in machine %d is:", n);
    int bestendline;
    bestendline=oppath[n-1];
    printf("%d\n", time[bestendline-1][n-1]);
    for(i=0; i<n; i++)
    {
        printf("%d.) Machine %d of Line %d\n", i+1, i+1, oppath[i]);
    }
    printf("\n");
    printf("Press any key to Continue...");
    getch();
    system("cls");
}

int inputchoice(int count, int ch1)
{
    int ch;
    char x;
    if(count<4)
    {
        printf("\nWhat do you want to input?? \n");
        printf("1. The time taken to process the product in each machine \n");
        printf("2. The entry time for each of the line \n");
        printf("3. The exit time for each of the line \n");
        printf("4. The time taken to travel from a machine in one level to the machine in the other level\n");
        printf("5. To reset all the inputs\n");
        printf("6. To start over\n\n");
        printf("Enter the choice: ");
        scanf("%d", &ch);
        printf("\n");
        return ch;
    }
    else
    {
        printf("You have entered data successfully...");
```

DSA.c

```c
        printf("\n\n");
        printf("Double ENTER to calculate the optimised path... ");
        getch();
        scanf("%c", &x);
        system("cls");
        return 5;
    }
}

void input(int e1[], int e2[], int **w1, int ***w2)
{
    int flag=0, flag1=0, flag2=0, flag3=0;
    char x;
    int i,j,k;
    int ch=0;
    int count=0;
    ch=inputchoice(count,ch);
    while(count<4)
    {
        switch(ch)
        {
            case 1:
                if(flag==1)
                {
                    printf("You have already input this data!\n");
                }
                else
                {
                    flag=1;
                    count=count+1;
                    printf("========================================================================================================================\n\n");
                    for(i=0; i<lines; i++)
                    {
                        printf("Lines %d\n", i+1);
                        for(j=0; j<machines; j++)
                        {
```

DSA.c

```c
                    printf("    Machine %d: ", j+1);
                    scanf("%d", &w1[i][j]);
                    printf("\n");
                }
            }
        }
        printf("\n");
        printf("Press any key to Continue...");
        getch();
        scanf("%c", &x);
        system("cls");
        ch=inputchoice(count,ch);
        break;

case 2:
        if(flag1==1)
        {
            printf("You have already input this data!\n");
        }
        else
        {
            flag1=1;
            count=count+1;
            printf("=======================================================================================================================\n\n");
            for(i=0; i<lines; i++)
            {
                printf("Line %d:", i+1);
                scanf("%d", &e1[i]);
                printf("\n\n");
            }
        }
        printf("\n");
        printf("Press any key to Continue...");
        getch();
        scanf("%c", &x);
        system("cls");
```

DSA.c

```c
        ch=inputchoice(count,ch);
        break;
case 3:
        if(flag2==1)
        {
            printf("You have already input this data!\n");
        }
        else
        {
            flag2=1;
            count=count+1;
            printf("=======================================================================================================================\n\n");
            for(i=0; i<lines; i++)
            {
                printf("Line %d:", i+1);
                scanf("%d", &e2[i]);
                printf("\n\n");
            }
        }
        printf("\n");
        printf("Press any key to Continue...");
        getch();
        scanf("%c", &x);
        system("cls");
        ch=inputchoice(count,ch);
        break;
case 4:
        if(flag3==1)
        {
            printf("You have already input this data!\n");
        }
        else
        {
            flag3=1;
            count=count+1;
            printf("=======================================================================================================================\n\n");
```

DSA.c

```c
                    for(i=0; i<lines; i++)
                    {
                        for(j=0; j<machines-1; j++)
                        {
                            printf("The time taken to travel from Machine %d of Line %d to: \n", j+1, i+1);
                            for(k=0; k<lines; k++)
                            {
                                printf("Machine %d of line %d: \n", j+2, k+1);
                                scanf("%d", &w2[i][j][k]);
                            }
                        }
                    }
                }
                printf("\n");
                printf("Press any key to Continue...");
                getch();
                scanf("%c", &x);
                system("cls");
                ch=inputchoice(count,ch);
                break;
            case 5:
                flag=0;
                flag1=0;
                flag2=0;
                flag3=0;
                count=0;
                check=0;
                printf("\n");
                printf("The values haven been reset!\n\n");
                printf("Press any key to Continue...");
                getch();
                scanf("%c", &x);
                system("cls");
                ch=inputchoice(count,ch);
                break;
            case 6:
```

DSA.c

```c
                system("cls");
                count=0;
                check=0;
                main();
                break;
            default:
                printf("Invalid input\n");
                printf("Press any key to Try again...");
                getch();
                scanf("%c", &x);
                system("cls");
                ch=inputchoice(count,ch);
                break;
            }
        }
        check=count;
}

int main()
{
    int c=0, ch;
    printf("\t\t\tHere we go!!\n\n");
    char x;
    printf("\tFirst enter the number of lines and machines in your factory...\n");
    printf("\tThe number of lines: ");
    scanf("%d", &lines);
    printf("\tThe number of machines in each line:");
    scanf("%d", &machines);
    printf("\n\n");
    printf("Press any key to Continue... ");
    getch();
    scanf("%c", &x);
    system("cls");
    int i,j,k;
    int e1[lines];
    int e2[lines];
```

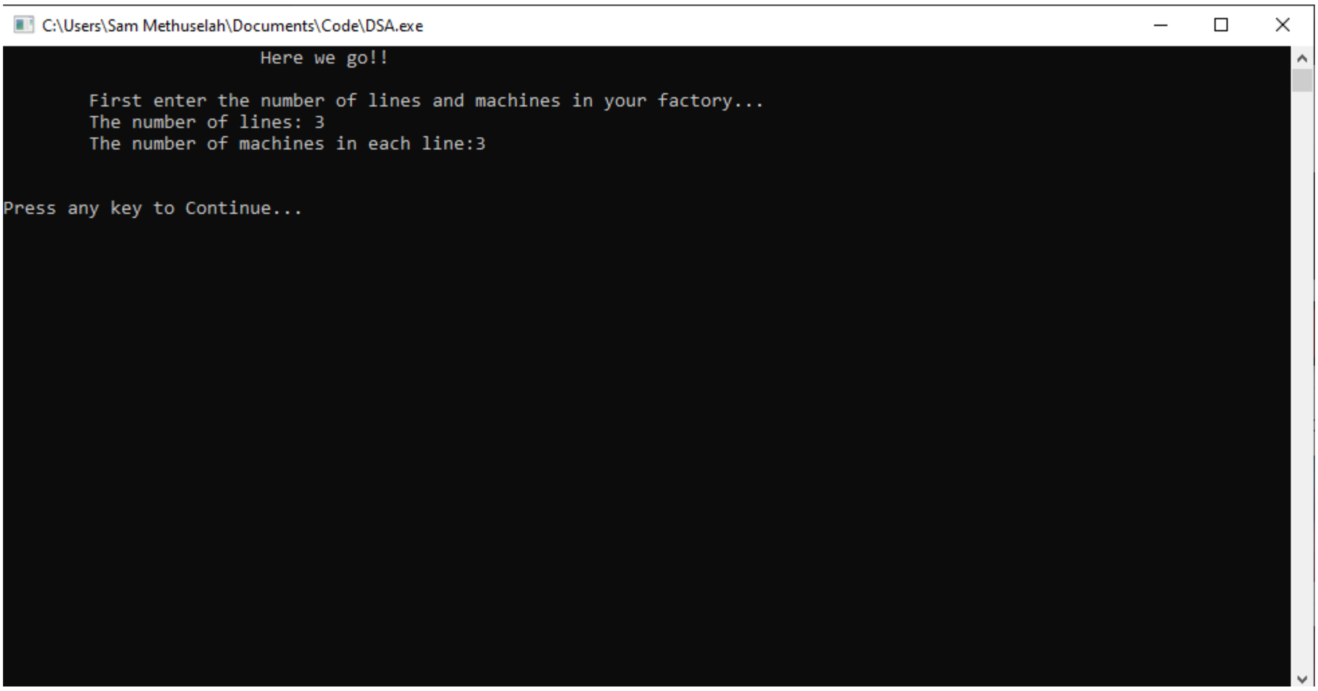
DSA.c

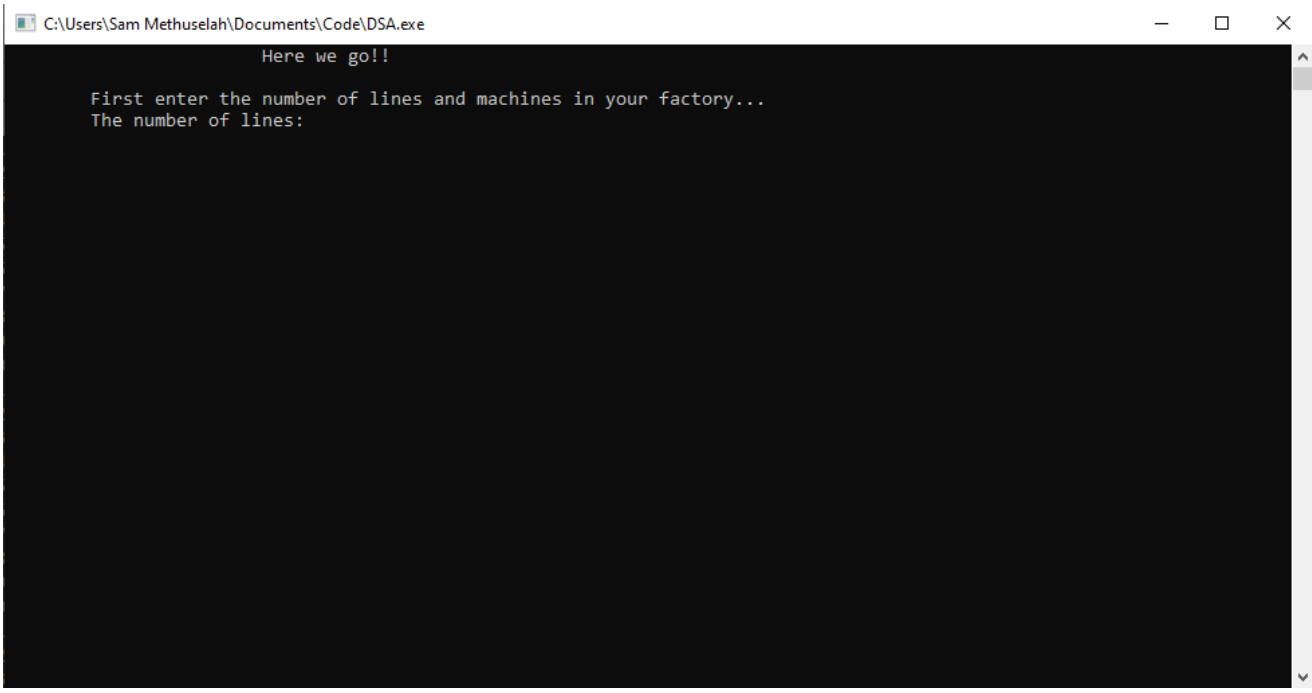
```c

    int *w1[lines];
    for (i=0; i<lines; i++)
    {
        w1[i]= (int *)malloc(machines* sizeof(int));
    }

    int ***w2= (int***)malloc(lines *sizeof(int**));
    for(i=0; i<lines; i++)
    {
        w2[i]=(int**)malloc(machines*sizeof(int*));
        for(j=0; j<machines; j++)
        {
            w2[i][j]=(int*)malloc(machines* sizeof(int));
        }
    }
    again:
    printf("                              HOME PAGE:\n");
    printf("\nPress 1 to Input the data");
    printf("\t\t\t\t(PLEASE INPUT THE DATA FIRST!!! IF DONE, IGNORE THIS MESSAGE)");
    printf("\nPress 2 to Input Menu");
    printf("\nPress 3 to Display the input data");
    printf("\nPress 4 to Process and Display the most optimised path");
    printf("\nPress 5 to END the program");
    printf("\n\nEnter the choice:");
    scanf("%d", &ch);
    system("cls");
    switch(ch)
    {
        case 1:
            input(e1,e2,w1,w2);
            process(e1,e2,w1,w2);
            goto again;
            break;
        case 2:
            input(e1,e2,w1,w2);
```

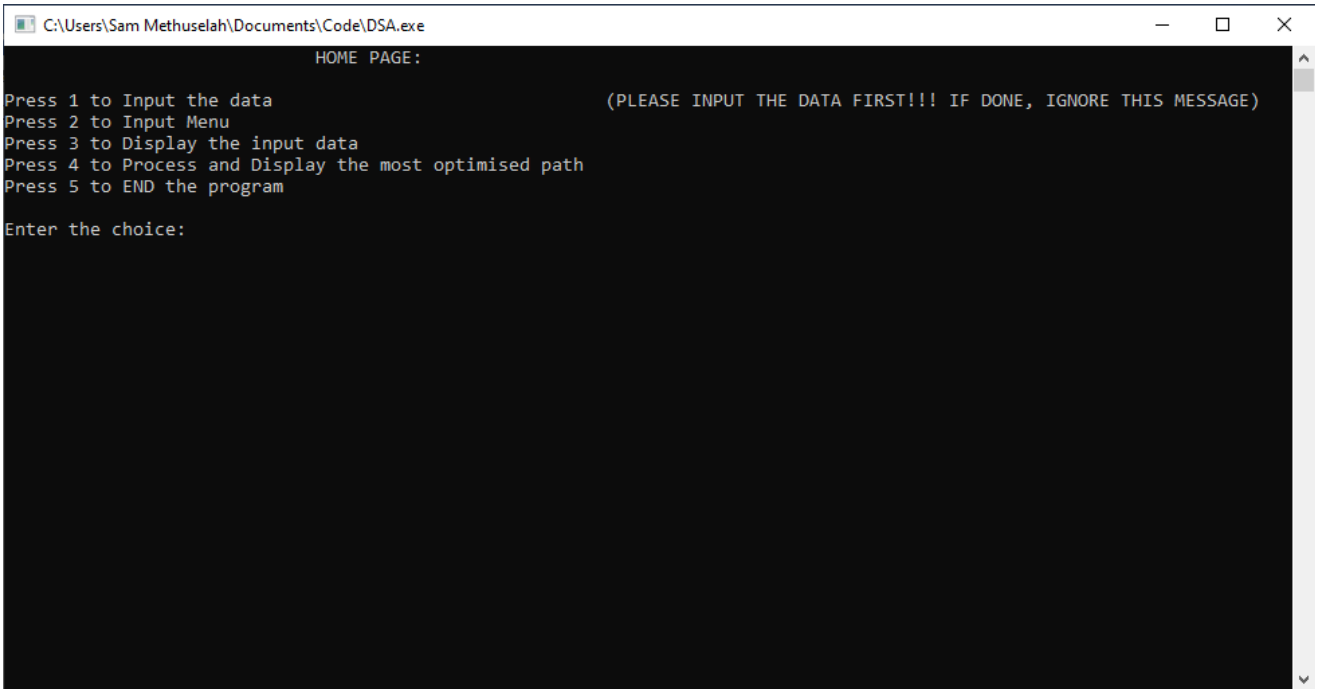
DSA.c

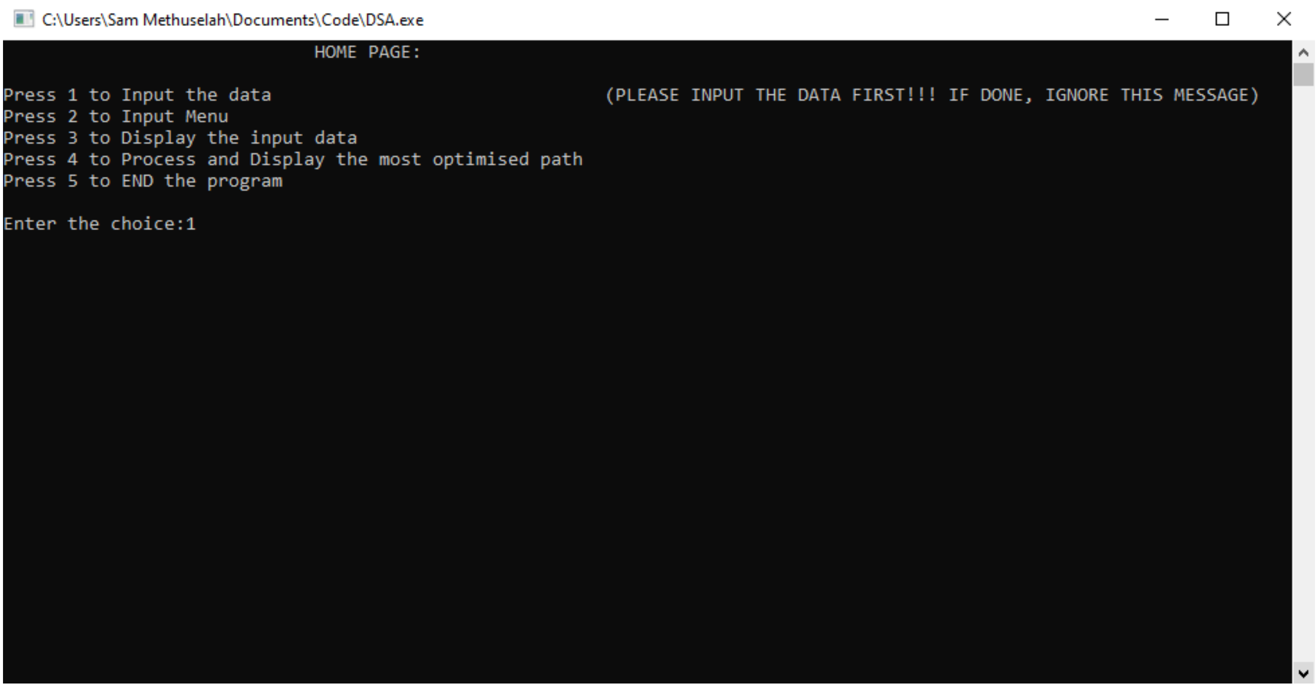
```c
    system("cls");
    switch(ch)
    {
        case 1:
            input(e1,e2,w1,w2);
            process(e1,e2,w1,w2);
            goto again;
            break;
        case 2:
            input(e1,e2,w1,w2);
            goto again;
            break;
        case 3:
            display(e1,e2,w1,w2);
            goto again;
            break;
        case 4:
            process(e1,e2,w1,w2);
            goto again;
            break;
        case 5:
            exit(0);
            break;
        default:
            printf("\nWRONG CHOICE!!!");
            break;
    }
    return 0;
}
```

SNAPSHOTS (Output)-






C:\Users\Sam Methuselah\Documents\Code\DSA.exe
HOME PAGE:
Press 1 to Input the data
(PLEASE INPUT THE DATA FIRST!!! IF DONE, IGNORE THIS MESSAGE)
Press 2 to Input Menu
Press 3 to Display the input data
Press 4 to Process and Display the most optimised path
Press 5 to END the program
Enter the choice:


C:\Users\Sam Methuselah\Documents\Code\DSA.exe
HOME PAGE:
Press 1 to Input the data
(PLEASE INPUT THE DATA FIRST!!! IF DONE, IGNORE THIS MESSAGE)
Press 2 to Input Menu
Press 3 to Display the input data
Press 4 to Process and Display the most optimised path
Press 5 to END the program
Enter the choice:1

```
C:\Users\Sam Methuselah\Documents\Code\DSA.exe

What do you want to input??
1. The time taken to process the product in each machine
2. The entry time for each of the line
3. The exit time for each of the line
4. The time taken to travel from a machine in one level to the machine in the other level
5. To reset all the inputs
6. To start over

Enter the choice:
```

```
C:\Users\Sam Methuselah\Documents\Code\DSA.exe

What do you want to input??
1. The time taken to process the product in each machine
2. The entry time for each of the line
3. The exit time for each of the line
4. The time taken to travel from a machine in one level to the machine in the other level
5. To reset all the inputs
6. To start over

Enter the choice: 1

========================================================================================================================
Lines 1
        Machine 1: 1

        Machine 2: 2

        Machine 3: 3

Lines 2
        Machine 1: 4

        Machine 2: 5

        Machine 3: 6

Lines 3
        Machine 1: 3

        Machine 2: 4

        Machine 3: 1


Press any key to Continue...
```

```
C:\Users\Sam Methuselah\Documents\Code\DSA.exe

What do you want to input??
1. The time taken to process the product in each machine
2. The entry time for each of the line
3. The exit time for each of the line
4. The time taken to travel from a machine in one level to the machine in the other level
5. To reset all the inputs
6. To start over

Enter the choice: 2

========================================================================================================================

Line 1:5


Line 2:2


Line 3:7



Press any key to Continue...
```

```
C:\Users\Sam Methuselah\Documents\Code\DSA.exe

What do you want to input??
1. The time taken to process the product in each machine
2. The entry time for each of the line
3. The exit time for each of the line
4. The time taken to travel from a machine in one level to the machine in the other level
5. To reset all the inputs
6. To start over

Enter the choice: 3

========================================================================================================================

Line 1:3


Line 2:2


Line 3:1



Press any key to Continue...
```

C:\Users\Sam Methuselah\Documents\Code\DSA.exe

```
What do you want to input??
1. The time taken to process the product in each machine
2. The entry time for each of the line
3. The exit time for each of the line
4. The time taken to travel from a machine in one level to the machine in the other level
5. To reset all the inputs
6. To start over

Enter the choice: 4

========================================================================================================================

The time taken to travel from Machine 1 of Line 1 to:
Machine 2 of line 1:
1
Machine 2 of line 2:
2
Machine 2 of line 3:
9
The time taken to travel from Machine 2 of Line 1 to:
Machine 3 of line 1:
2
Machine 3 of line 2:
3
Machine 3 of line 3:
7
The time taken to travel from Machine 1 of Line 2 to:
Machine 2 of line 1:
4
Machine 2 of line 2:
8
Machine 2 of line 3:
2
The time taken to travel from Machine 2 of Line 2 to:
Machine 3 of line 1:
3
Machine 3 of line 2:
```

C:\Users\Sam Methuselah\Documents\Code\DSA.exe

```
Machine 3 of line 1:
2
Machine 3 of line 2:
3
Machine 3 of line 3:
7
The time taken to travel from Machine 1 of Line 2 to:
Machine 2 of line 1:
4
Machine 2 of line 2:
8
Machine 2 of line 3:
2
The time taken to travel from Machine 2 of Line 2 to:
Machine 3 of line 1:
3
Machine 3 of line 2:
4
Machine 3 of line 3:
5
The time taken to travel from Machine 1 of Line 3 to:
Machine 2 of line 1:
4
Machine 2 of line 2:
3
Machine 2 of line 3:
1
The time taken to travel from Machine 2 of Line 3 to:
Machine 3 of line 1:
0
Machine 3 of line 2:
2
Machine 3 of line 3:
5

Press any key to Continue...
```

```
C:\Users\Sam Methuselah\Documents\Code\DSA.exe
You have entered data successfully...

Double ENTER to calculate the optimised path...
```

Optimized Path-

```
C:\Users\Sam Methuselah\Documents\Code\DSA.exe
The least time required for a product to be made is : 17

1.) Machine 1 of Line 1
2.) Machine 2 of Line 1
3.) Machine 3 of Line 1


Press any key to return to the HOME page...
```

RESULTS

Algorithm Analysis and Comparison:

Dynamic programming recurrences do (often) consider all possible ways to split the given problem instance into smaller instances according to some scheme. However, it will not combine all solutions to all partial problems with each other and pick the best -- it combines only optimal partial solutions (and picks the best out of those).

Brute forcing, on the other hand, is trial and error. It does not utilize any intelligent approach towards the problem, rather takes the simplest logic and repeats it continuously until the solution is arrived.

Now, comparing it to the other two optimization paradigm, namely, Divide and Conquer, and the Greedy Algorithm. Like divide-and-conquer, dynamic programming results optimal solutions by combining the partial best possible solutions to sub-problems. Unlike the case in divide-and-conquer algorithms, immediate implementation of the recurrence results in identical recursive calls that are executed more than once. The structure of dynamic programming is similar to divide-and-conquer, except that the subproblems to be solved are overlapping in nature which makes as a consequence different recursive path to the same subproblems. Thus, for solving a problem, divide- and-conquers are Independent sub-problems, solve sub-problems independently and recursively. Conversely, in dynamic programming sub- problems are dependent.

Greedy method is also a powerful technique for optimizations but not much like dynamic programming approach. In greedy, we solve a problem making greedy choices. After the choice is made the subproblem is arising. These choices may depend on previous choices. However, the choice is independent of the solutions to subproblems. Top-down convention is normally used towards the feasible solution decreasing current problem size. Unlike greedy, choice is made at each step and bottom up approach is employed increasing problem size from smaller to larger subproblems answering optimal solutions. It is more powerful than greedy as it could be applicable to wide range of applications.

COMPARISON

vs BRUTE FORCING

Brute forcing is trial and error. It does not utilize any intelligent approach towards the problem, rather takes the simplest logic and repeats it continuously until the solution is arrived.

vs DIVIDE AND CONQUER

In a greedy Algorithm, we make whatever choice seems best at the moment in the hope that it will lead to global optimal solution. The greedy method computes its solution by making its choices in a serial forward fashion, never looking back or revising previous choices

vs GREEDY ALGORITHM

In greedy, we solve a problem making greedy choices. After the choice is made the subproblem is arising. These choices may depend on previous choices. However, the choice is independent of the solutions to subproblems. Top-down convention is normally used towards the feasible solution decreasing current problem size. Unlike greedy, choice is made at each step and bottom up approach is employed increasing problem size from smaller to larger subproblems answering optimal solutions. It is more powerful than greedy as it could be applicable to wide range of applications.

CONCLUSION

Algorithm Complexity:

The complexity of the Brute Force method utilized in solving the Assembly Line Scheduling Problem is $O(m^n)$. Here, m is the number of Assembly Lines given in the problem, and n is the number of stations.

Now, the algorithm used in our project is the Dynamic Algorithm. The complexity of our solution is $O(m^2n)$. Here, m is the number of Assembly lines and n is the number of stations given in the problem.

Advantages:

- The process of breaking down a complex problem into a series of interrelated sub problems often provides insight into the nature of problem. In our code, we found the optimized time taken to reach each station and then summed it up, hence, making it easier to get to the optimized solution.
- Dynamic programming achieves computational savings over complete enumeration.
- The computational procedure in dynamic programming allows for a built-in form of sensitivity analysis based on state variables and on variables represented by stages.

Limitation:

- The major shortcoming of making use of dynamic programming as a means is that it is often nontrivial to write code that evaluates the subproblems in the most efficient order.
- Another drawback of this practice is that it works best on objects which are linearly ordered and cannot be rearranged such as characters in a string, points around the boundary of a polygon, matrices in a chain, the left-to-right order of leaves in a search tree.
- The major shortcoming of making use of dynamic programming as a means is that it is often nontrivial to write code that evaluates the subproblems in the most efficient order.

REFERENCES

- www.academia.edu
- www.sanfoundry.com
- www.nhu.edu.tw